E.U.I.T. TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA
PLAN 2000

TEMA: Comunicaciones Móviles

TÍTULO: "Sistema de control de temperatura a través de Arduino y la tecnología GPRS/GSM"

AUTOR: Alberto Castro Domínguez

TUTOR: Rafael Herradón Díez **Vº Bº.**

DEPARTAMENTO: DIAC

Miembros del Tribunal Calificador:

PRESIDENTE: Javier Luis Palmero Huerta

VOCAL: Rafael Herradón Díez

VOCAL SECRETARIO: Florentino Jiménez Muñoz

DIRECTOR:

Fecha de lectura:

Calificación: **El Secretario,**

RESUMEN DEL PROYECTO:

El objetivo del proyecto consiste en estudiar las posibilidades de desarrollo de un sistema para el control de la temperatura basado en la plataforma Arduino. Con el fin de alcanzar dicho objetivo, se implementará un sistema que permita la consulta y control de la temperatura ambiente a través de la red de comunicaciones móviles.

Tras un análisis previo de las distintas placas Arduino, se evaluarán una serie de módulos de expansión (shields) compatibles con dicha plataforma que nos permiten ampliar sus funcionalidades dotando al dispositivo de un sistema de comunicación basado en la tecnología GPRS/GSM.

Se estudiarán los diferentes sensores de temperatura compatibles con Arduino, además de una serie de actuadores que contribuyan al accionamiento y control de un termostato, y al desarrollo de un pequeño sistema de alarma frente a detección de temperaturas extremas.

Concluiremos con el diseño de una aplicación basada en el entorno de desarrollo Arduino que nos permita evaluar las distintas capacidades de nuestro sistema, así como comunicarnos con la plataforma a través de SMS para el control remoto de la temperatura.

Resumen:

El principal objetivo de este proyecto consiste en estudiar las posibilidades de desarrollo de un sistema para el control de la temperatura basado en la plataforma Arduino. Con el fin de alcanzar dicho objetivo, se ha implementado un sistema que permite la consulta y control de la temperatura ambiente a través de la red de comunicaciones móviles.

Tras un análisis previo de las distintas placas Arduino, se evalúan una serie de módulos de expansión (shields) compatibles con dicha plataforma que nos permiten ampliar sus funcionalidades, dotando al dispositivo de un sistema de comunicación basado en la tecnología GPRS/GSM.

Se estudian los diferentes sensores de temperatura compatibles con Arduino, además de una serie de actuadores que contribuyen al accionamiento y control de un posible termostato, así como al desarrollo de un pequeño sistema de alarma capaz de detectar temperaturas extremas.

El proyecto concluye con el diseño de una aplicación basada en el entorno de desarrollo Arduino que nos permita evaluar las distintas capacidades de nuestro sistema, así como comunicarnos con la plataforma a través de SMS para el control remoto de la temperatura.

Abstract:

The goal of the project consists of studying the developmental possibilities of a temperature control system based on the Arduino platform. In order to this, there has been implemented a system to consult and manage the environmental temperature through mobile communication networks.

After a previous assessment of the different Arduino boards, there are analysed a set of expansion modules (shields) compatibles with the platform that enables us to upgrade the device functionalities with the GPRS/GSM communication protocol.

Different temperature sensors compatible with Arduino have been studied. In addition, there are evaluated a set of actuators for the operation and control of a thermostat and also the development of a small alarm system that alerts of extremes temperatures.

The project concludes with the design of an application based on the Arduino development environment which allows us to evaluate the different capabilities of our system as well as the communication with the platform by SMS for the remote temperature control.

Agradecimientos:

A mis padres, Emilio y Pilar, por todo su apoyo y comprensión, por estar siempre ahí, en los buenos y en los malos momentos, y por haberme dado la oportunidad de tener unos estudios. Sin vosotros nunca hubiera logrado la obtención de este título.

A mi hermana Belén, cuya constancia en sus estudios me ha servido de influencia para coger fuerzas en los momentos más difíciles y sacar el trabajo adelante.

Por supuesto a mi novia, Berta, por haber estado a mi lado todos estos años, por aquellos buenos momentos que hemos compartido y compartiremos. Sin tu motivación nada de esto hubiera sido posible.

Y gracias también a todos aquellos que de una forma u otra hayan contribuido a la consecución de esta meta.

Gracias a todos.

ÍNDICE

ÍNDICE DE FIGURAS 3

1. INTRODUCCIÓN 7

1.1. MOTIVACIÓN DEL PROYECTO 7

1.2. PLANTEAMIENTO Y OBJETIVOS DEL PROYECTO 8

1.3. SUMARIO DE LOS CONTENIDOS DEL PROYECTO 11

2. ESTADO DEL ARTE 12

2.1. EL INTERNET DE LAS COSAS 12

2.2. GSM 14

2.3. GPRS 16

2.4. COMANDOS AT 17

3. LA PLATAFORMA ARDUINO 20

3.1. PLACAS ARDUINO 21

3.2. ENTORNO DE DESARROLLO EN ARDUINO 27

3.3. SHIELDS GPRS/GSM 31

3.4. SENSORES DE TEMPERATURA 37

4. DESARROLLO EXPERIMENTAL 42

4.1. INSTALANDO EL AMBIENTE DE DESARROLLO DE ARDUINO 42

4.2. COMENZANDO CON ARDUINO. HOLA MUNDO 43

4.3. EVALUACIÓN DE LOS DISTINTOS SENSORES DE TEMPERATURA 47

4.4. ACCIONAR UN SERVOMOTOR 54

4.5. SISTEMA DE ALARMA CON UN BUZZER Y UN LED 58

4.6. SHIELD GPRS/GSM 60

4.6.1. Configuración previa del módulo 61

4.6.2. Envío de SMS por comandos AT 64

4.6.3. Envío de SMS a través de Arduino 68

4.6.4. Lectura de SMS por comandos AT 71

4.6.5. Lectura de SMS a través de Arduino 74

4.7. DESARROLLO DE APLICACIONES Y EVALUACIÓN DEL SISTEMA 79

4.7.1. Código de la aplicación general para el sistema de control de temperatura de gestión local 80

4.7.2. Evaluación del sistema de gestión local 90

4.7.3. Código de aplicación para la gestión remota de la temperatura. Modo 'standalone' 95

4.7.4. Evaluación del sistema de gestión remota 101

5. CONCLUSIONES 104

6. ANEXO_A: REFERENCIAS 106

ÍNDICE DE FIGURAS

Figura 1.1: Mapa de cobertura de la red ADSL en España 9

Figura 1.2: Mapa de cobertura de la red 3G en España 9

Figura 2.1: Concepto del "Internet de las cosas" 12

Figura 2.2: Expectativas de crecimiento de dispositivos inteligentes interconectados 13

Figura 2.3: Arquitectura de red del sistema GSM 15

Figura 2.4: Arquitectura de red del sistema GPRS 17

Figura 3.1: Fotografía de la placa Arduino UNO 20

Figura 3.2: Arduino Duemilanove 21

Figura 3.3: Arduino UNO 22

Figura 3.4: Arduino Leonardo 22

Figura 3.5: Arduino MEGA 23

Figura 3.6: Arduino Fio 23

Figura 3.7: Arduino Nano 24

Figura 3.8: Arduino LilyPad 24

Figura 3.9: Arduino Bluetooth 24

Figura 3.10: Arduino Ethernet 25

Figura 3.11: Arduino Mini 25

Figura 3.12: Arduino Pro Mini 25

Figura 3.13: Arduino Pro 26

Figura 3.14: Arduino Robot 26

Figura 3.15: Arduino Esplora 27

Figura 3.16: Entorno de desarrollo en Arduino 28

Figura 3.17: Selección de caracteres de fin de línea en la ventana "Monitor Serial" 30

Figura 3.18: Selección del valor de 'baudrate' en la ventana "Monitor Serial 30

Figura 3.19: Ejemplo de estructura modular con varias shields para Arduino 31

Figura 3.20: Módulo GPRS Quadband para Arduino.....32

Figura 3.21: Diagrama de puertos y conexiones del módulo GPRS Quadband 32

Figura 3.22: Módulo GPRS+GPS Quadband para Arduino y Raspberry PI (SIM 908).....33

Figura 3.23: Diagrama de conexiones del módulo GPRS+GPS Quadband (SIM908).....34

Figura 3.24: Módulo 3G/GPRS+GPS para Arduino/Raspberry PI.....35

Figura 3.25: Diagrama de puertos y conexiones del módulo 3G/GPRS+GPS 35

Figura 3.26: Módulo GPRS/GSM Quadband para Arduino (SIM900) 36

Figura 3.27: Diagrama de conexiones del módulo GPRS/GSM Quadband (SIM900).....36

Figura 3.28: Sensor de temperatura DS18B20.....38

Figura 3.29: Esquema de montaje para la prueba del sensor DS18B20 38

Figura 3.30: Sensor de temperatura TMP36 39

Figura 3.31: Esquema de montaje para la prueba del sensor TMP36.....39

Figura 3.32: Sensor de temperatura SEN118A2B 40

Figura 3.33: Esquema de montaje para la prueba del sensor SEN118A2B 41

Figura 4.1: Selección del puerto serie en el IDE de Arduino para MAC.....42

Figura 4.2: Ventana para la selección del modelo de placa de Arduino 43

Figura 4.3: Esquema de montaje para hacer parpadear un LED con Arduino.....44

Figura 4.4: Selección de códigos de ejemplo incluidos en el IDE de Arduino.....45

Figura 4.5: Código correspondiente al sketch Blink.....46

Figura 4.6: Proceso de carga del sketch en Arduino.....46

Figura 4.7: Montaje del sensor de temperatura TMP36 sobre la plataforma Arduino UNO..47

Figura 4.8: Resultado de la compilación del sketch....49

Figura 4.9: Monitor Serial durante la ejecución del programa de prueba del sensor TMP36 50

Figura 4.10: Montaje del sensor de temperatura SEN118A2B sobre la plataforma Arduino. 51

Figura 4.11: Monitor Serial durante la prueba del sensor SEN118A2B....53

Figura 4.12: Micro Servo 9G....54

Figura 4.13: Esquema de montaje de motor servo....55

Figura 4.14: Montaje de motor servo sobre la plataforma Arduino UNO....55

Figura 4.15: Importación de librerías en el IDE de Arduino....56

Figura 4.16: Código correspondiente al sketch de prueba de un motor servo....57

Figura 4.17: Esquema de montaje para el sistema de alarma con un Buzzer y un LED....58

Figura 4.18: Sketch correspondiente a un posible sistema de alarma....59

Figura 4.19: Módulo GPRS/GSM (SIM900)....60

Figura 4.20: Diagrama de pines y conexiones del módulo GPRS/GSM (SIM900)....62

Figura 4.21: Plataforma Arduino UNO + Shield GPRS/GSM....64

Figura 4.22: Colocación de los jumpers en modo "USB gateway"....65

Figura 4.23: Código para la configuración del módulo GPRS/GSM en modo gateway....65

Figura 4.24: Ventana de selección de caracteres de fin de línea....66

Figura 4.25: Monitor Serial durante el intercambio de comandos AT para el envío de SMS 66

Figura 4.26: Ventana de selección de caracteres de fin de línea....67

Figura 4.27: Esquema de montaje para el envío de SMS a través de Arduino....70

Figura 4.28: Recepción del SMS en el teléfono móvil....70

Figura 4.29: Sketch para la configuración del módulo en modo gateway....71

Figura 4.30: Intercambio de comandos AT para la lectura de un SMS....73

Figura 4.31: Nueva shield GPRS/GSM (SIM900)....74

Figura 4.32: Jumper para la selección de modo Arduino/Raspberry Pi 75

Figura 4.33: Sketch correspondiente al código para lectura de un SMS 77

Figura 4.34: Colocación de los jumpers operando en modo "USB gateway" 78

Figura 4.35: Conexión de la plataforma con los jumpers en modo Arduino 78

Figura 4.36: Monitor Serial tras la lectura de un SMS a través de Arduino 79

Figura 4.37: Comparativa entre los modelos Arduino UNO y Arduino MEGA 81

Figura 4.38: Diagrama de flujo de la aplicación general 82

Figura 4.39: Montaje correspondiente al prototipo del sistema completo 91

Figura 4.40: Menú de inicio 91

Figura 4.41: Consultar el valor de temperatura 92

Figura 4.42: Enviar SMS con el valor de temperatura 93

Figura 4.43: SMS recibido con el valor de temperatura 93

Figura 4.44: Activar alarma por temperatura extrema 94

Figura 4.45: SMS recibido tras el accionamiento del sistema de alarma 94

Figura 4.46: Desactivar el sistema de alarma 95

Figura 4.47: Fijar la temperatura del termostato 95

Figura 4.48: Diagrama de flujo correspondiente al sistema de gestión remota 96

Figura 4.49: Montaje del sistema en modo 'standalone' 101

Figura 4.50: Consultar temperatura por SMS 102

Figura 4.51: Ajustar la temperatura del termostato por SMS 103

Figura 5.1: Presupuesto total del sistema 104

1. INTRODUCCIÓN

1.1. MOTIVACIÓN DEL PROYECTO

Todo comenzó con el descubrimiento de la plataforma Arduino allá por Septiembre de 2012. Gracias a un compañero, conocí por primera vez la existencia de Arduino. Me lo presentó como una plataforma de hardware libre basada en una placa que constaba de un microcontrolador y un sencillo entorno de desarrollo, gracias al cual te permitía implementar de manera muy simple infinidad de proyectos relacionados con las telecomunicaciones.

Desde entonces me pico la curiosidad, y comencé a investigar sobre las posibilidades que ofrecía dicha plataforma. Rápidamente me di cuenta de que era cierta la fama con la que contaba el ahora ya conocido Arduino. Al tratarse de una plataforma de hardware libre, en internet abundaba la información relacionada con este dispositivo, y comencé a interesarme de verdad por algunas de las funcionalidades que podrían resultar de la utilización de ésta interesante herramienta.

Tras observar la gran variedad de placas con las que contaba Arduino, y las distintas Shields, sensores y actuadores compatibles con ellas, hubo una que llamó especialmente mi atención. Se trataba de una Shield basada en un módulo SIM900 compatible con la mayoría de placas Arduino, y a través de la cual podíamos convertir un simple dispositivo en una especie de terminal móvil, capaz de integrar una tarjeta SIM y establecer comunicación a través de las tecnologías GSM/GPRS con otros dispositivos móviles.

Entonces fue cuando comencé a plantearme seriamente enfocar mi Proyecto Fin de Carrera hacia la plataforma Arduino. La idea inicial consistía en diseñar una especie de plataforma de control de la temperatura a través de la tecnología GSM/GPRS, e implementar una aplicación Android que fuese capaz de comunicarse con Arduino y controlar remotamente la plataforma.

Tras comentarlo con mi compañero, decidimos plantear la idea a nuestro profesor de la asignatura Comunicaciones Móviles, a quién creímos que le podría resultar interesante ser nuestro tutor para este proyecto. La reunión fue todo éxito, y salimos completamente convencidos de que Arduino sería la base de nuestro Proyecto Fin de Carrera.

1.2. PLANTEAMIENTO Y OBJETIVOS DEL PROYECTO

Como se introducía en el apartado anterior, la idea de partida para el desarrollo del proyecto se basaba en el diseño de una plataforma de control de la temperatura por medio de una aplicación móvil. La intención desde un principio era llegar a programar nuestra propia aplicación Android, a través de la cual pudiésemos llegar a controlar a distancia la plataforma Arduino, que a su vez, se encargaría de la gestión de una red de sensores y actuadores capaces de detectar la temperatura ambiente en todo momento, regular un termostato, incluso activar una serie de alarmas en función del valor de temperatura detectado.

El tipo de tecnología que utilizaríamos para la comunicación entre el dispositivo móvil y la plataforma Arduino siempre estuvo un poco en el aire. Se barajaban tres posibilidades: Wi-Fi, GPRS/GSM, o una combinación de las mismas. Tras evaluar las distintas shields compatibles con Arduino, vimos que cualquiera de las tres opciones era posible.

Una de las premisas que fijamos desde un principio fue tratar de reducir al máximo los costes del posible dispositivo, de manera que obtuviésemos como resultado una herramienta de bajo coste que estuviese al alcance de un gran público. Además, sería necesario que la gran mayoría de la población (posibles usuarios) contase con los medios necesarios para la utilización de dicho dispositivo.

Tras evaluar los costes que conllevaría la elección de cada una de las tecnologías planteadas y la posibilidad de que los usuarios tuviesen a su disposición los medios necesarios para contar con esas tecnologías, siempre cobró más fuerza la idea de utilizar la shield GPRS/GSM para la comunicación con Arduino. De este modo, a pesar de que el coste en hardware no suponía una gran diferencia respecto a decantarnos por el uso de Wi-Fi, sí lo hacía el desembolso necesario para disponer de acceso a internet en determinadas circunstancias para las cuales estaba siendo diseñado el proyecto. Por ejemplo, una de las ideas para la creación de este dispositivo era poder instalarlo en una vivienda en la que normalmente el propietario no residiese, es decir, que muy probablemente no se tratase de una gran ciudad, sino más bien de una zona rural, donde al propietario le interesase mantener bajo control la temperatura de su domicilio o segunda vivienda.

A pesar de que nos cueste creerlo, la cobertura de las conexiones ADSL todavía es un tema pendiente en nuestro país. Si bien casi todos los hogares disponen de línea telefónica, no todas las líneas permiten ofrecer este servicio. Además, las exigencias de calidad del ADSL, tanto de ruido, como de atenuación, son bastante más críticas que para el servicio telefónico básico, lo que sigue frenando en gran medida la expansión del acceso a internet por todo el territorio. En la figura 1.1 podemos ver el mapa de cobertura de la red ADSL en España. [1]

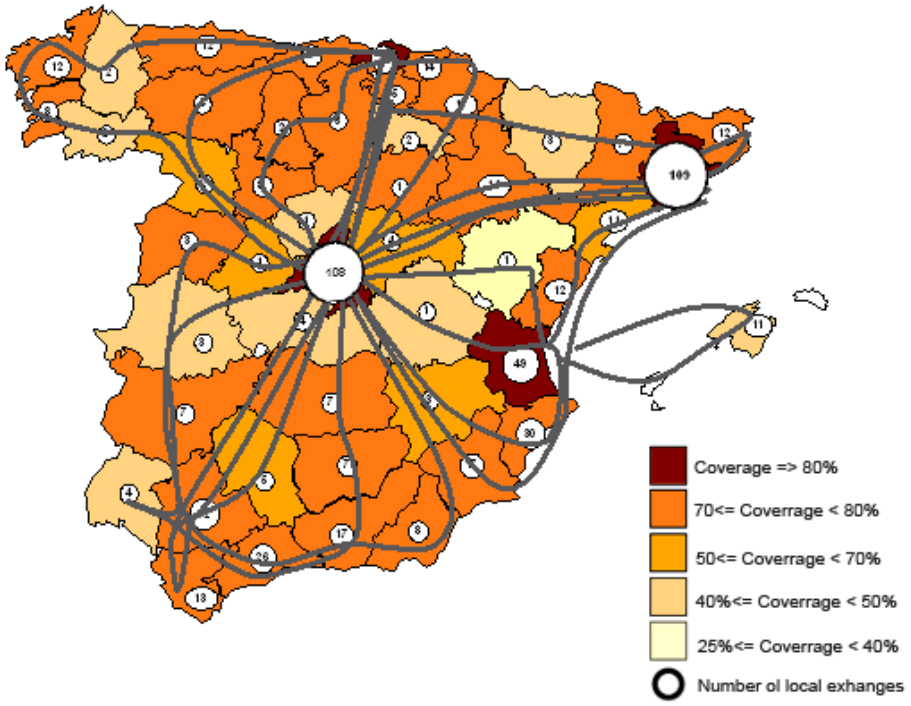


Figura 1.1: Mapa de cobertura de la red ADSL en España

Sin embargo, como podemos observar en la figura 1.2, la red de telefonía móvil dispone de una cobertura más extensa a lo largo de todo nuestro territorio, lo que inclinó la balanza a su favor, e hizo que finalmente nos decantásemos por utilizar esta tecnología. [2]

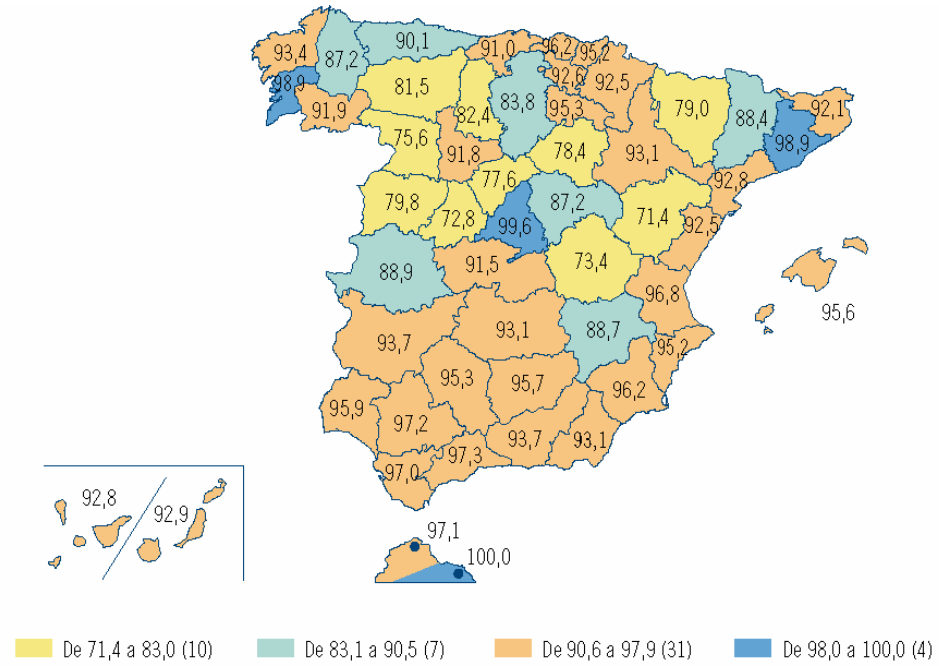


Figura 1.2: Mapa de cobertura de la red 3G en España

En todo proyecto, las cosas nunca salen como se plantean desde un inicio, y siempre hay que ir adaptándose a los posibles contratiempos que puedan ir surgiendo. En nuestro caso, no iba a ser menos, y tras el desarrollo de este último curso, mi compañero y yo nos vimos forzados a dividir el proyecto. Tuvimos que centrarnos en ver cómo podíamos separarlo en dos fases, de manera que pudiésemos presentar cada uno una parte de lo que se pretendía que fuese el proyecto en todo su conjunto.

Como resultado surgió un nuevo planteamiento que comprendería dos fases; una primera fase en la que se diseñe la plataforma para el control de temperatura a modo local con el apoyo de un PC y el envío/recepción de SMS para el control de todo el sistema; y una segunda fase en la que se integre una aplicación Android para la gestión de estas mismas funcionalidades.

Este contratiempo conlleva por tanto un nuevo enfoque a la hora de encarar el desarrollo del proyecto, el cual decido abordarlo en varias etapas:

1) Familiarización con la plataforma Arduino.
2) Integración de la red de sensores y actuadores en la plataforma.
3) Evaluación de la Shield GPRS/GSM y sus posibilidades de comunicación.
4) Interconexión del sistema global. Montaje de la plataforma.

El objetivo de esta primera fase se resume por tanto en el diseño de una plataforma compuesta por la placa Arduino, una shield GPRS/GSM, un sensor de temperatura, y una serie de actuadores, capaces de llevar a cabo ciertas funciones como por ejemplo: la consulta de la temperatura ambiente, la activación/desactivación de un pequeño sistema de alarma frente a temperaturas límite, el control de un termostato mediante el accionamiento de un servomotor, y el envío de SMS con el valor de temperatura o con alertas por exceso de la misma. A su vez, se pretende implementar otra aplicación que sea capaz de trabajar de manera desatendida (‘standalone’), y que responda ante mensajes de texto para consultar el valor de la temperatura en cualquier momento o regular el termostato acorde a las instrucciones que el usuario le transmita vía SMS.

Todos estos objetivos serán siempre con vistas al desarrollo de una mejora del sistema que llevará a cabo mi compañero en la segunda fase de este proyecto, en la que se desarrollará una aplicación Android para móvil o Tablet con el fin de hacer más dinámica la gestión de todo el sistema, además de incluir mejoras en las prestaciones que pueda ofrecer la plataforma.

1.3. SUMARIO DE LOS CONTENIDOS DEL PROYECTO

El proyecto se divide en tres grandes apartados que abarcarán las diferentes fases por las que se ha tenido que pasar para el desarrollo del mismo.

En primer lugar, llevamos a cabo un pequeño estudio teórico sobre el estado del arte de las diferentes tecnologías que guardan alguna relación con el proyecto, como son el concepto de Internet de las Cosas, los sistemas de comunicaciones móviles GSM y GPRS, y los principales comandos AT. Todo ello quedará resumido en el capítulo 2.

Tras ello, entraremos en el apartado 3, en el que se evaluará todo lo relacionado con la base del proyecto, es decir, con la plataforma Arduino (placas, shields de extensión, sensores, actuadores, etc.). Comenzaremos analizando los diferentes modelos de placas Arduino disponibles en el mercado, con el fin de escoger el que mejor se adapte a las necesidades del sistema que se tiene como objetivo desarrollar. A continuación, examinaremos los módulos de expansión (shields) compatibles con esta plataforma, que nos permitan extender sus capacidades y contar con un dispositivo capaz de establecer un medio de comunicación a través de la tecnología GPRS/GSM. Sin dejar este mismo apartado, presentaremos los distintos sensores de temperatura que hemos podido encontrar compatibles con Arduino.

En el siguiente apartado, capítulo 4, se expondrán con detalle todas las tareas realizadas en el desarrollo experimental del proyecto, como las pruebas con Arduino, análisis de los sensores de temperatura, pruebas de envío y recepción de SMS a través del módulo GPRS/GSM, etc.

Por último, dejaremos un apartado para expresar las conclusiones que se han podido sacar tras el resultado final del proyecto.

2. ESTADO DEL ARTE

En este capítulo se va a realizar un resumen sobre el estado del arte de las diferentes tecnologías implicadas en el desarrollo del proyecto, las cuales podremos asociar con el término 'internet de las cosas'. Se trata de los sistemas de comunicaciones GSM, GPRS, y la utilización de Comandos Hayes (o comandos AT).

2.1. EL INTERNET DE LAS COSAS

La expresión "Internet de las Cosas" hace referencia a la interconexión entre objetos de consumo o de uso cotidiano (electrodomésticos, ropa, libros, productos alimenticios, etc.) a través de ciertos dispositivos capaces de conectarlos a la red. En la figura 2.1 se resume el concepto de IoT.

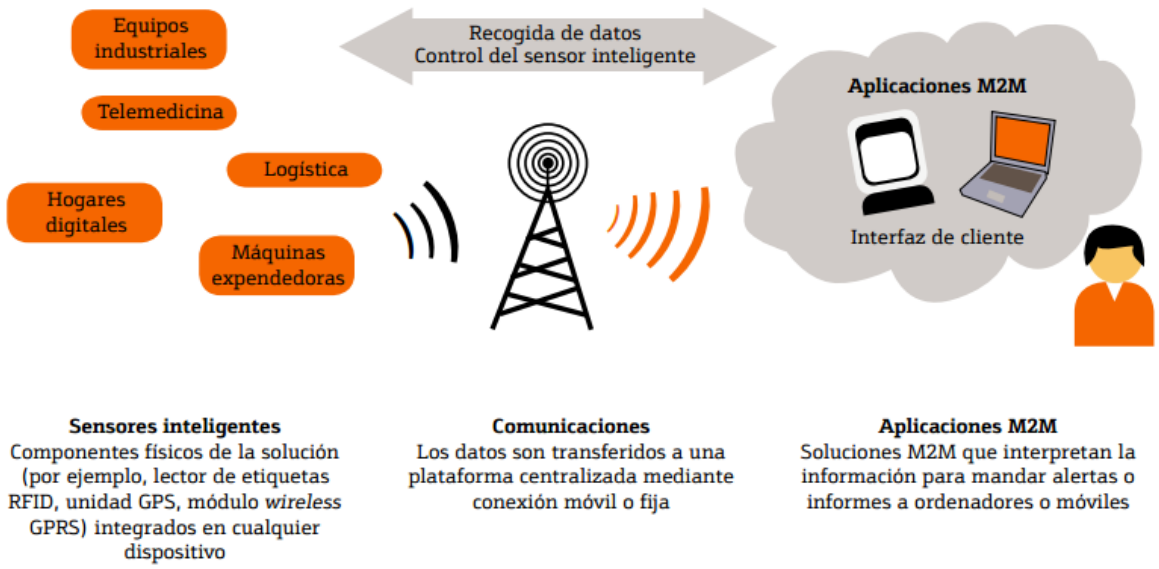


Figura 2.1: Concepto del "Internet de las cosas"

Las utilidades del Internet de las Cosas pueden considerarse prácticamente infinitas. Cada vez son más los medios que tenemos a nuestra disposición para implementar, de manera no muy compleja, un pos ible dispositivo capaz de dotar a cierto objeto o f unción de una conexión a la red, ya sea por medio de una tecnología u otra.

La llegada de IPv6 supone un factor clave en el desarrollo del concepto IoT. Gracias a este nuevo protocolo (diseñado para reemplazar a IPv4) se evitará que el crecimiento de Internet quede restringido, y hará posible la gestión de direccionamiento de innumerables dispositivos.

Por otro lado, ya son muchas las aplicaciones móviles y servicios en la nube que nos permiten la conexión a todos estos dispositivos, y proporcionan una vía para el tratamiento de una inmensa cantidad de datos en tiempo real (sistemas 'Big Data'), facilitando la integración de infinidad de sensores aplicables prácticamente a cualquier tipo de necesidad. De hecho, ya han surgido incluso redes sociales de sensores, como la plataforma 'Xively' [3], donde los usuarios comparten datos en tiempo real procedentes de distintos sensores.

En los próximos años se espera un gran aumento en el número de equipos de uso cotidiano interconectados, entre otras cosas, gracias a la inminente llegada de todos estos sensores inteligentes a nuestros hogares. En la figura 2.2 se muestra un gráfico con las expectativas de crecimiento en vistas al año 2025 [4].

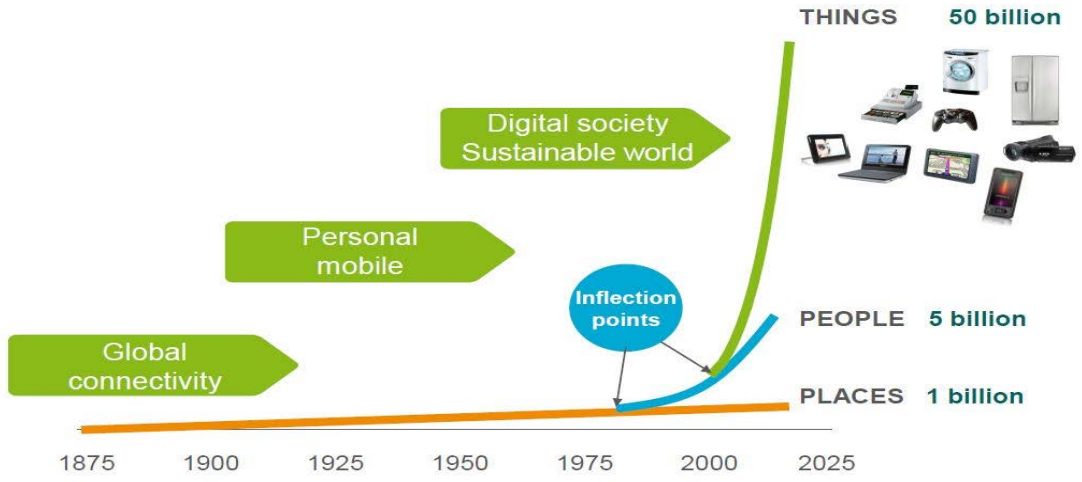


Figura 2.2: Expectativas de crecimiento de dispositivos inteligentes interconectados

También son muchas las empresas que ofrecen, o están interesadas, en soluciones IoT. La gestión de recursos y la eficiencia energética son las aplicaciones más solicitas. Tecnologías inalámbricas como las redes móviles, WIFI, Zigbee, Bluetooth, etc., permiten optimizar posibles soluciones y facilitan su despliegue.

Sin ir más lejos, Arduino ha marcado un punto de inflexión en este sentido, convirtiéndose en la herramienta ideal para llevar a cabo multitud de prototipos y obtener nuevos caos de uso. Gracias a su bajo coste, sencillez y a la variedad de modelos que podemos encontrar, resulta una herramienta de gran ayuda a la hora de implementar ideas y soluciones de ámbito doméstico. Además, existen multitud de sensores y actuadores compatibles con esta plataforma, mediante los cuales podemos recopilar datos de nuestro entorno, analizarlos, y actuar en consecuencia, incluso conectar con otros dispositivos a través de las distintas tecnologías de comunicación (GSM/GPRS, 3G, Bluetooth, RFID, etc.) aprovechando toda una variedad de shields de expansión.

2.2. GSM

GSM es la abreviatura de 'Sistema Global para las comunicaciones Móviles' (en inglés, Global System for Mobile communications). A comienzos del siglo XXI, es el estándar más utilizado de Europa. [5] Conocido como estándar de segunda generación (2G), su principal diferencia respecto a la primera generación de teléfonos móviles es que sus comunicaciones son totalmente digitales.

El estándar GSM fue desarrollado a partir de 1982, cuando fue estandarizado por primera vez, denominado "Groupe Spécial Mobile". Surgió como idea para el desarrollo de un estándar europeo de telefonía móvil digital. En 1991 se convirtió en un estándar internacional llamado "Sistema Global de Comunicaciones Móviles", y comenzaron a presentarse los primeros prototipos de telefonía GSM.

En Europa, el sistema GSM utiliza las bandas de frecuencia de 850, 900 y 1800 MHz, mientras que en los Estados Unidos se usa la banda de frecuencia de 1900 MHz. En consecuencia, los dispositivos de comunicaciones móviles que pueden operar tanto en Europa como en Estados Unidos se conocen como cuatribanda (Quadband).

El estándar GSM permite transmisiones digitales de voz y datos, como mensajes de texto (SMS) o mensajes multimedia (MMS).

Respecto a su arquitectura de red, en GSM todo terminal móvil debe estar constituido por una tarjeta SIM (Módulo de identificación de abonado) y el propio dispositivo, normalmente un teléfono móvil. [6]

La tarjeta SIM es la encargada de identificar en la red al usuario y al terminal móvil. Estos dispositivos se identifican gracias a un número exclusivo de identificación denominado IMEI (Identificador internacional de equipos móviles), compuesto por 15 dígitos. Por otro lado, cada tarjeta SIM también posee un número de identificación único denominado IMSI (Identificador internacional de abonados móviles).

En la figura 2.2 podemos ver la arquitectura de red correspondiente al sistema GSM. Está compuesta por múltiples estaciones base (BTS), que a su vez, se conectan a un controlador de estaciones base (BSC), encargado de la administración de la red. A éste sistema compuesto por el BSC y sus correspondientes estaciones base conectadas al mismo, se le conoce como BSS (Subsistema de estaciones base).

En un nivel superior estarían los Centros de conmutación móvil (MSC), al que se conectan físicamente los controladores de estaciones base. Éste es el encargado de establecer la conexión con la red de telefonía pública y con Internet. Su administración corre a cargo del

operador de la red telefónica. El MSC pertenece a un Subsistema de conmutación de red (NSS), el cual se encarga de identificar a los usuarios, determinar su ubicación, y gestionar las comunicaciones con otros usuarios de la red. A su vez, el Centro de Conmutación móvil (MSC) se conecta a una serie de base de datos que le proporcionan funciones adicionales:

- Registro de posición base (HLR): en esta base de datos se almacena la información de los abonados (posición geográfica, información administrativa, etc.).

- Registro de posición visitante (VLR): contiene información de usuarios que no son abonados locales. Los datos se conservan mientras el usuario está dentro de la zona, y se eliminan en cuanto abandona la zona o tras un largo período de inactividad.

- Registro de identificación del equipo (EIR): es una base de datos que contiene la lista con los dispositivos móviles.

- El Centro de autenticación (AUC): su misión es verificar las identidades de los usuarios.

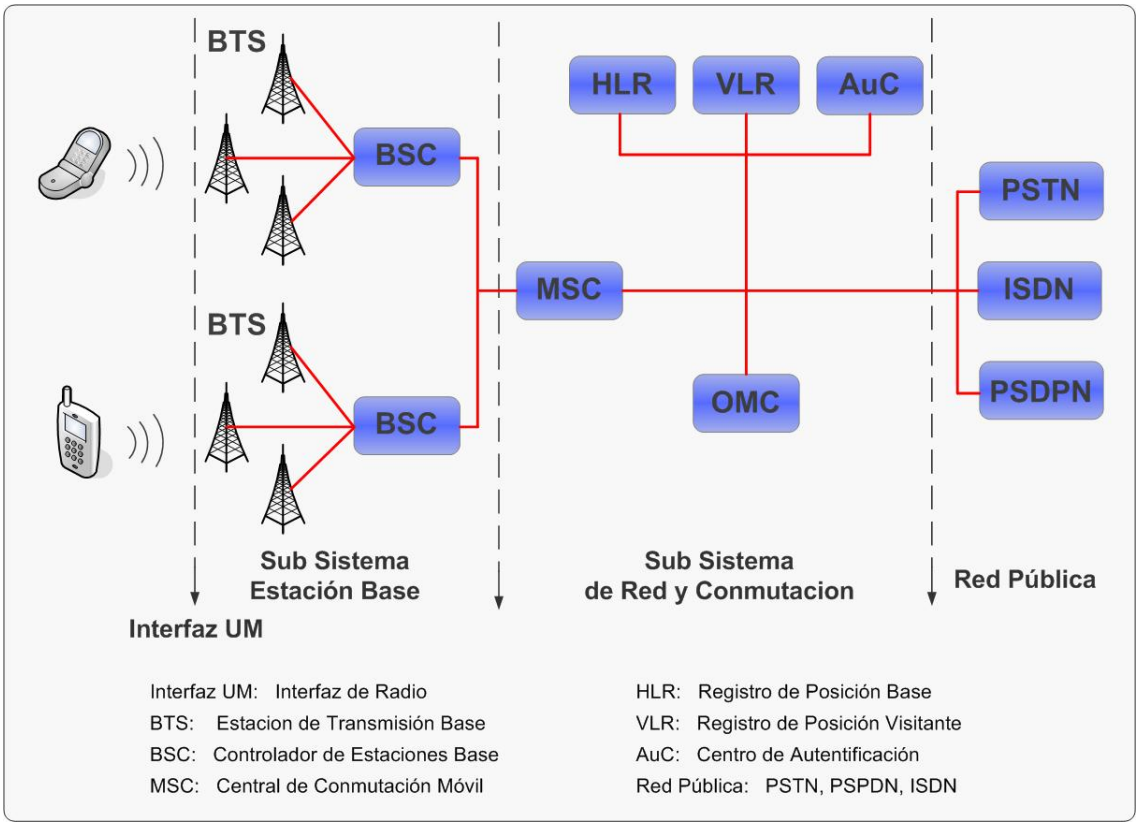


Figura 2.3: Arquitectura de red del sistema GSM

2.3. GPRS

El estándar GPRS o Servicio General de Paquetes vía Radio (en inglés, General Packet Radio Service) es una evolución del sistema GSM. Es también conocido como GSM++, pero dado que se trata de un estándar de telefonía móvil intermedio entre la segunda generación (2G) y la tercera (3G), a menudo recibe la nomenclatura de 2.5G. [7]

GPRS extiende la arquitectura del estándar GSM para permitir la transferencia de datos mediante conmutación de paquetes con velocidades de transferencia que rondan los 114 Kbps.

Al contrario de lo que ocurre en conmutación de circuitos, en el estándar GPRS, gracias a su modo de transferencia de paquetes, las transmisiones de datos sólo utilizan la red cuando es necesario, permitiendo la tarificación por volumen de información transmitida en lugar de por tiempo de conexión, por lo tanto, el usuario puede permanecer conectado sin costo adicional, ya que sólo utilizará la red cuando envíe o reciba un paquete de datos.

Para el acceso a la red de datos, el estándar GPRS utiliza el protocolo IP, mientras que para el transporte de voz, emplea la arquitectura de la red GSM.

A parte de actualizar algunos servicios con los que ya contaba GSM, la tecnología GPRS admite otra serie de características que no estaban disponibles en 2G:

- Servicios de mensajes cortos (SMS)
- Servicios de mensajes multimedia (MMS)
- Servicio punto a punto (PTP); para la conexión cliente-servidor en una red IP
- Servicio punto a multipunto (PTMP); para el envío de multidifusión.

En la figura 2.3 se muestra la estructura funcional del sistema GPRS, basada en la adición de nuevos nodos sobre la infraestructura correspondiente a GSM. A dichos nodos se les conoce como GSN (nodos de soporte GPRS):

- El SGSN (Nodo de soporte de servicios GPRS) se encarga del encaminamiento de los paquetes IP entrantes y salientes dirigidos hacia, o procedentes, de cualquier abonado GPRS situado dentro de la zona geográfica a la que da servicio ese SGSN.

- EL GGSN (Nodo de soporte pasarela GPRS) sirve de interfaz hacia las redes externas de paquetes IP. Encamina las direcciones IP de los abonados servidos por la red GPRS, intercambiando información de encaminamiento con la red externa. El GGSN prepara la comunicación con redes externas y gestiona las sesiones de GPRS. También incluye funciones para asociar abonados con la SGSN que corresponda. También recopila datos de tarificación y uso de la red de datos externa.

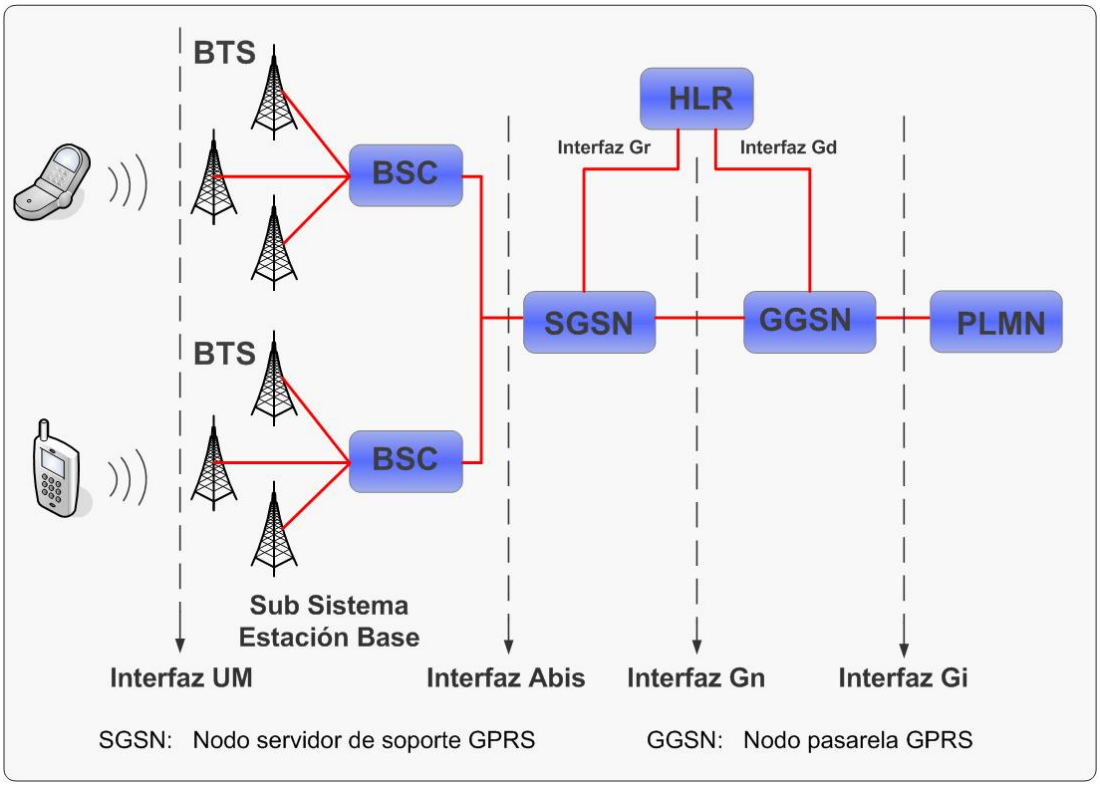


Figura 2.4: Arquitectura de red del sistema GPRS

2.4. COMANDOS AT

Los comandos AT, también conocidos como comandos Hayes (en honor a su desarrollador Dennis Hayes), son una serie de instrucciones que conforman un interfaz de comunicación entre usuario y modem. Su abreviatura AT por la que son mundialmente conocidos estos comandos proviene de la palabra 'attention'.

Aunque la finalidad principal de los comandos AT fue la comunicación con módems, la telefonía móvil GSM/GPRS también adoptó este lenguaje como estándar de comunicación.

En la actualidad, todos los terminales móviles GSM poseen una serie específica de comandos AT que nos permiten configurarlos por medio de estas instrucciones e indicarles una serie de acciones que queremos que ejecuten, tales como marcar un número de teléfono, enviar o leer un SMS, consultar el estado de conexión a la red, leer o escribir en la agenda de contactos, etc.

Gracias a que la transmisión de comandos AT no depende del canal de comunicación a través del cual estos sean enviados (cable, infrarrojos, Bluetooth, etc.), podremos utilizar

nuestra placa Arduino para transmitir dichos comandos a un módulo GPRS/GSM que sea capaz de interpretarlos y actuar en consecuencia.

Como son muchos los comandos existentes (véase el manual de comandos AT oficial de nuestro módulo GSM/GPRS (SIM900) [8]), únicamente vamos a mencionar junto con una breve descripción, aquellos que puedan resultarnos de principal interés en el desarrollo del proyecto:

- AT – Con este comando verificaremos que el módulo está recibiendo nuestras instrucciones. Si todo es correcto, tras enviarlo debe responder con un "OK". En caso contrario no obtendremos respuesta alguna.

- AT+CPIN? – Utilizaremos esta instrucción para comprobar si nuestra tarjeta SIM está desbloqueada o si por el contrario, requiere introducir el código PIN para proceder con el desbloqueo de la misma. Cuando la SIM esté operativa responderá con un "ready".

- AT+CPIN="****" – En el caso de que necesitemos introducir el PIN, éste es el comando que debemos enviar, escribiendo los 4 dígitos correspondientes al código de desbloqueo en el lugar de los asteriscos, delimitado entre comillas.

- AT+CREG? – Con este comando estamos preguntando por el estado de conexión a la red. La respuesta recibida seguirá la siguiente notación: +CREG: <n>,<stat>, donde:

  <stat> = 0 → No registrado, no está buscando una conexión de red
  <stat> = 1 → Registrado a la red nacional
  <stat> = 2 → No registrado, pero buscando la red
  <stat> = 3 → Registro denegado
  <stat> = 5 → Registro tipo roaming

- AT+COPS? – Mediante esta instrucción recibiremos la confirmación de la compañía en la que está registrada nuestra tarjeta SIM (Movistar, Orange, Vodafone, etc.)

- AT+CMGF=<f> – Selecciona el formato del mensaje SMS, donde:
  <f> = 0 → modo PDU
  <f> = 1 → modo texto

- AT+CSCS="IRA" – Con este comando seleccionamos el set de caracteres que es utilizado para el envío de SMS. En nuestro caso nos interesa configurar el módulo en modo "IRA" (International Reference Alphabet).

- AT+CMGS=”*num_movil*” – A través de este comando marcaremos el número de móvil al que queremos hacer llegar el SMS. Irá delimitado por comillas.

- AT+CMGR=<r> – Es el comando utilizado para leer SMS almacenados en la memoria de la tarjeta SIM. En lugar de <r> pondremos el número correspondiente a la posición del mensaje que queremos leer. Por ejemplo, si queremos leer el último mensaje recibido, pondremos un ‘ 1’, si queremos leer el penúltimo mensaje pondremos un ‘2’, y así sucesivamente.

- AT+CPMS=”SM” – Con esta instrucción seleccionamos el directorio del que queremos leer un mensaje de texto. En función de las siglas que pongamos tras el igual, podemos recopilar la información de distintas memorias. En este caso, con las siglas “SM”, seleccionamos como directorio la memoria de la tarjeta SIM.

- AT+CMGD=<n> – Este comando sirve para eliminar SMS de la memoria de la tarjeta SIM. Sustituiremos <n> por la posición en la memoria que ocupe el SMS que queremos borrar. Por ejemplo, para borrar el primer mensaje almacenado en la memoria de la SIM la instrucción sería AT+CMGD=1.

- AT+CMGL=”ALL” – Con este comando podemos ver en una lista todos los SMS que tengamos en la SIM, tanto leídos, como no leídos.

- +CMTI: “SM”, <pos> – Esta es la instrucción que recibiremos automáticamente por parte del módulo cuando se reciba un SMS, donde <pos> es el número correspondiente a la posición en memoria en la que se ha almacenado dicho mensaje. Por ejemplo, si tenemos la memoria de la SIM vacía y nos llega un SMS, la instrucción que nos enviaría el módulo sería +CMTI: “SM”, 1.

NOTA: Para el envío de cualquiera de los comandos AT a través de Hyperterminal o del entorno de desarrollo de Arduino, debemos seleccionar siempre los caracteres fin de línea y retorno de carro.

3. LA PLATAFORMA ARDUINO

A continuación entramos en detalle sobre los distintos elementos que conformarán el sistema basado en la plataforma Arduino. Comenzaremos con una breve presentación de lo que es en sí esta plataforma, y analizaremos los diferentes modelos de placas disponibles en el mercado. También se evaluarán las distintas shields (módulos de expansión) GSM/GPRS compatibles con esta plataforma, y los principales sensores de temperatura que tenemos a nuestra disposición.

¿Qué es Arduino? - Arduino es una plataforma electrónica de hardware libre basada en una placa con un m icrocontrolador. Con software y hardware flexibles y fáciles de utilizar, Arduino ha sido diseñado para adaptarse a las necesidades de todo tipo de público, desde aficionados, hasta expertos en robótica o equipos electrónicos. También consta de un simple, pero completo, entorno de desarrollo, que nos permite interactuar con la plataforma de manera muy sencilla. Se puede definir por tanto como una sencilla herramienta de contribución a la creación de prototipos, entornos, u objetos interactivos destinados a proyectos multidisciplinares y multitecnología. [9] En la figura 3.1 podemos ver una de sus placas más vendidas a nivel mundial, se trata del modelo Arduino UNO.

Figura 3.1: Fotografía de la placa Arduino UNO

A través de Arduino podemos recopilar multitud de información del entorno sin excesiva complejidad. Gracias a sus pines de entrada, nos permite jugar con toda una gama de sensores (temperatura, luminosidad, presión, etc.) que nos brindan la capacidad de controlar o actuar sobre ciertos factores del entorno que le rodean, como por ejemplo: controlando luces, accionando motores, activando alarmas…y muchos otros actuadores.

Gracias a que la plataforma es de hardware libre, las placas Arduino pueden ser hechas a mano por cualquier aficionado o compradas ya montadas de fábrica.

Respecto al software, es totalmente gratuito, y está disponible para la descarga por cualquier interesado en la propia página web de Arduino [10]. El entorno de desarrollo dispone de un propio lenguaje de programación para el microcontrolador de la placa Arduino, basado en Processing/Wiring.

Una de los principales motivos por el cual resulta muy interesante la utilización de la plataforma Arduino para determinados proyectos, se basa en su independencia respecto a tener que mantenerse conectado a un PC. Arduino es perfectamente capaz de trabajar en modo 'standalone', solo es necesario asegurarnos de haber cargado previamente el programa que deseamos que mantenga en ejecución. Si bien, todo esto no le priva de poder operar manteniendo en todo momento la conexión con el PC, siendo capaz de comunicarse con diferentes tipos de software, como por ejemplo: Macromedia Flash, Processing, Max/MSP, Pure Data, etc.

3.1. PLACAS ARDUINO

Desde el momento de su creación, allá por el año 2005, cuando Arduino nació como un proyecto educativo, las innovaciones no han dejado de sucederse. [11] A día de hoy existen multitud de placas Arduino, y la mayoría de ellas están disponibles en distintas versiones, adaptables prácticamente a cualquier tipo de requisitos o necesidades para llevar a cabo un determinado proyecto. Los principales modelos de placas Arduino que podemos encontrar en el mercado a día de hoy son los siguientes:

- "Duemilanove".- Podemos decir que es la primera versión de la placa básica de Arduino capaz de seleccionar automáticamente la fuente de alimentación adecuada, USB o fuente externa, eliminando la necesidad de utilizar un jumper a modo de conmutador para la selección de una u otra opción, tal como ocurría en placas anteriores. Desde la aparición de Arduino UNO, la Duemilanove ha quedado en un segundo plano, pasando a ser un modelo obsoleto. Su precio está entorno a los 20€.

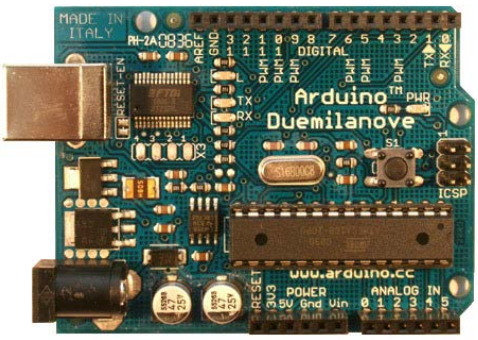


Figura 3.2: Arduino Duemilanove

- "UNO".- Es la última versión de la placa básica de Arduino. Como ya se ha comentado, ha sido la evolución de la Duemilanove. Se conecta al ordenador mediante un cable USB estándar y contiene todo lo necesario para comenzar a programar y utilizar la placa. Sus funcionalidades se pueden ver incrementadas gracias a que existen multitud de Shields perfectamente compatibles con este modelo. A diferencia de la antigua Duemilanove, integra un nuevo chip USB-serie y cuenta con un nuevo diseño de etiquetado, facilitando la identificación de las distintas entradas y salidas de la placa. Con un precio aproximado de 24€, se trata quizás de la placa Arduino más interesante a la hora de buscar la mejor relación precio-prestaciones, por lo que será uno de los modelos a tener muy en cuenta para el desarrollo del proyecto.

Figura 3.3: Arduino UNO

- "Leonardo".- Muy similar a Arduino UNO, se trata de una evolución del mismo, con mejores prestaciones y un precio similar. También es un modelo que puede resultar bastante interesante para el desarrollo de pequeños proyectos. Precio: 20€.

Figura 3.4: Arduino Leonardo

- "MEGA".- El Arduino Mega es probablemente la placa con mayores prestaciones de la familia Arduino. Cuenta con 54 pines digitales, que funcionan como entrada/salida, además de sus 16 entradas analógicas. Es la placa más grande y potente de Arduino. Es totalmente compatible con las shields Arduino UNO, y cuenta con una memoria que duplica su capacidad en comparación con el resto de placas. Arduino MEGA es por tanto la opción más adecuada para aquellos proyectos en los que se requiera un gran número de entradas y salidas disponibles, o pa ra soportar la carga de códigos de programación pesados que no pueden almacenarse en las memorias de menor capacidad que ofrecen otras placas, como por ejemplo el modelo UNO. Quizá por este motivo, resulte interesante contar con este modelo de placa para la realización del proyecto, ya que se pretenden utilizar bastantes entradas/salidas a las que conectar la Shield GSM/GPRS y los distintos sensores/actuadores para el desarrollo del proyecto. Además, el código puede llegar a ser bastante extenso cuando intentemos centralizar todas las funciones en una sola plataforma: control de temperatura, envío y recepción de SMS, activación de alarmas, etc. Su precio ronda los 45€, aunque podemos encontrar placas no oficiales por unos 25€.

Figura 3.5: Arduino MEGA

- "Fio" - Un Arduino orientado para su uso a modo de nodo inalámbrico. Posee conectores para la integración de un módulo XBee, y dispone de un conector para batería de litio y un circuito para cargar la batería. Precio 23€.

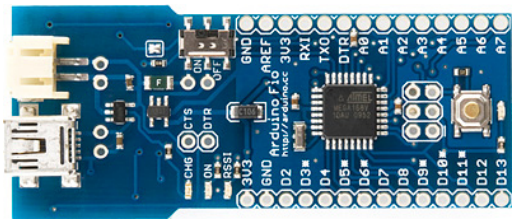


Figura 3.6: Arduino Fio

- "Nano" - Una placa compacta diseñada para usar directamente en placas de desarrollo. Su conexión con el PC debe ser a través de un cable Mini-B USB. Precio: 40€.

Figura 3.7: Arduino Nano

- "LilyPad" - Una modelo de la placa circular, de tamaño reducido, compacta y diseñada específicamente para ser cosida a la ropa o cualquier tipo de material flexible, orientado a cualquier tipo de aplicaciones. Necesita de un adaptador adicional para su comunicación con el PC. Precio: 20€.

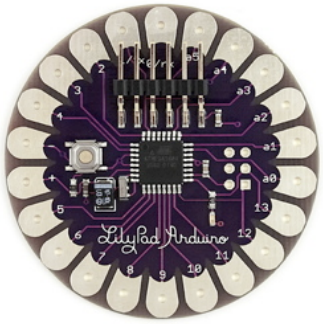


Figura 3.8: Arduino LilyPad

- "Bluetooth".- El Arduino BT contiene un módulo bluetooth que permite comunicarse y programarse sin necesidad de cableado.

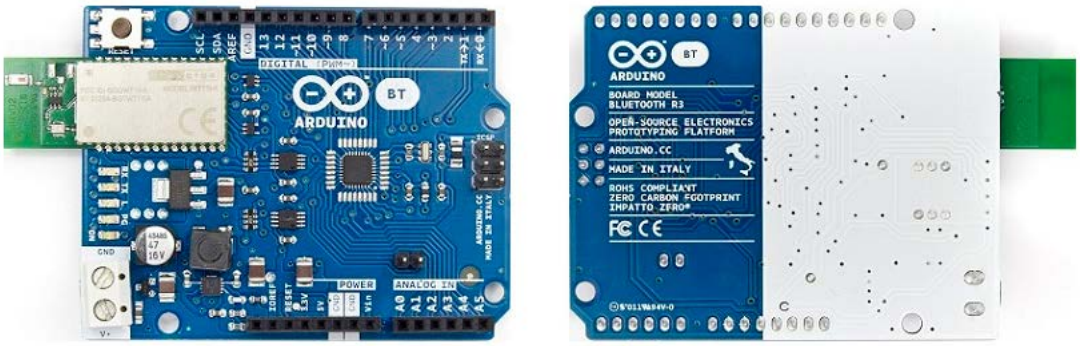


Figura 3.9: Arduino Bluetooth

- "Ethernet".- Simplemente se trata de una placa Arduino, del estilo a Leonardo o Arduino UNO, pero dotada con un puerto Ethernet para su conexión a Internet. Es una solución interesante para aquellos proyectos en los que se necesite estar conectado permanentemente a Internet. Precio: 35€.

Figura 3.10: Arduino Ethernet

- "Mini" – Es la placa Arduino más pequeña. Diseñada especialmente para aquellas aplicaciones en las que el espacio es primordial. Permite la conexión con un ordenador a través del adaptador Mini USB.

Figura 3.11: Arduino Mini

- "Pro Mini" - Como la Pro, la Pro Mini está diseñada para usuarios avanzados que buscan reducir los costes el máximo posible. Su tamaño es muy reducido. Precio: 23€

Figura 3.12: Arduino Pro Mini

- "Pro" - Placa muy interesante, diseñada para aquellos proyectos en los que es necesario dejar la placa instalada permanentemente. Es más barata que la Diecimila y se puede alimentar fácilmente con baterías. Precio: 22€.

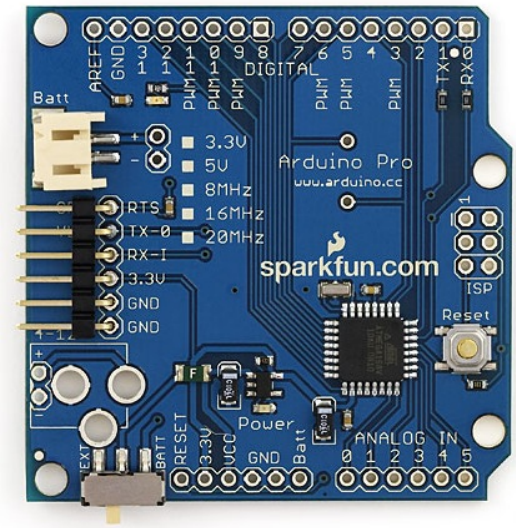


Figura 3.13: Arduino Pro

- "Robot".- Este innovador modelo es la primera placa oficial Arduino sobre ruedas. El robot consta de dos procesadores, uno para cada una de sus dos placas. Tiene una placa dedicada al control de los motores, y la otra es la encargada de procesar los datos recibidos por parte de los sensores y decidir cómo debe actuar en todo momento. El modelo Robot tiene muchos de sus pines ya asignados a los sensores y actuadores de a bordo, y su programación es muy similar al proceso requerido por el Arduino Leonardo. Queda claro que se trata de una placa destinada a la robótica. Precio: 200€.

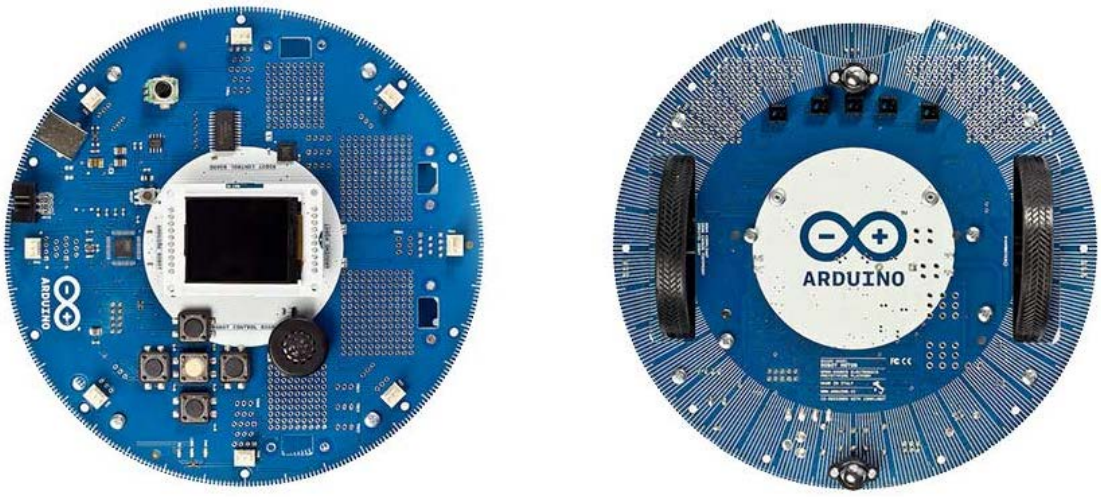


Figura 3.14: Arduino Robot

- "Esplora".- Es una placa derivada del modelo Leonardo, distinta a todas las demás placas Arduino precedentes. Preparada para utilizar sensores multitud de sensores a bordo, ha sido diseñada para aquellas personas que quieran empezar a jugar con Arduino sin tener que poseer ciertos conocimientos mínimos en electrónica. Esplora incorpora multitud de funcionalidades: un dispositivo capaz de emitir sonidos, un micrófono, sensores de luz, sensores de temperatura, incluso un acelerómetro y una palanca a modo de mando de control. Además, permite extender sus capacidades gracias a sus dos entradas TINKERKIT, conectores de salida, y puerto para la conexión de pantalla TFT LCD. Podemos encontrarla por unos 55€.

Figura 3.15: Arduino Esplora

3.2. ENTORNO DE DESARROLLO EN ARDUINO

El entorno de desarrollo en Arduino (IDE) es el encargado de la gestión de la conexión entre el PC y el hardware de Arduino con el fin de establecer una comunicación entre ellos por medio de la carga de programas. Como podemos ver en la figura 3.16, el IDE de Arduino se compone de: [12]

- Un **editor de texto**.- donde escribir el código del programa.

- Un **área de mensajes**.- a través del cual el usuario tendrá constancia en todo momento de los procesos que se encuentren en ejecución, errores en código, problemas de comunicación, etc.

- Una **consola de texto**.- mediante la que podremos comunicarnos con el hardware Arduino y viceversa.

- Una **barra de herramientas**.- donde podremos acceder a una serie de menús y a los botones con acceso directo a las principales funcionalidades de Arduino.

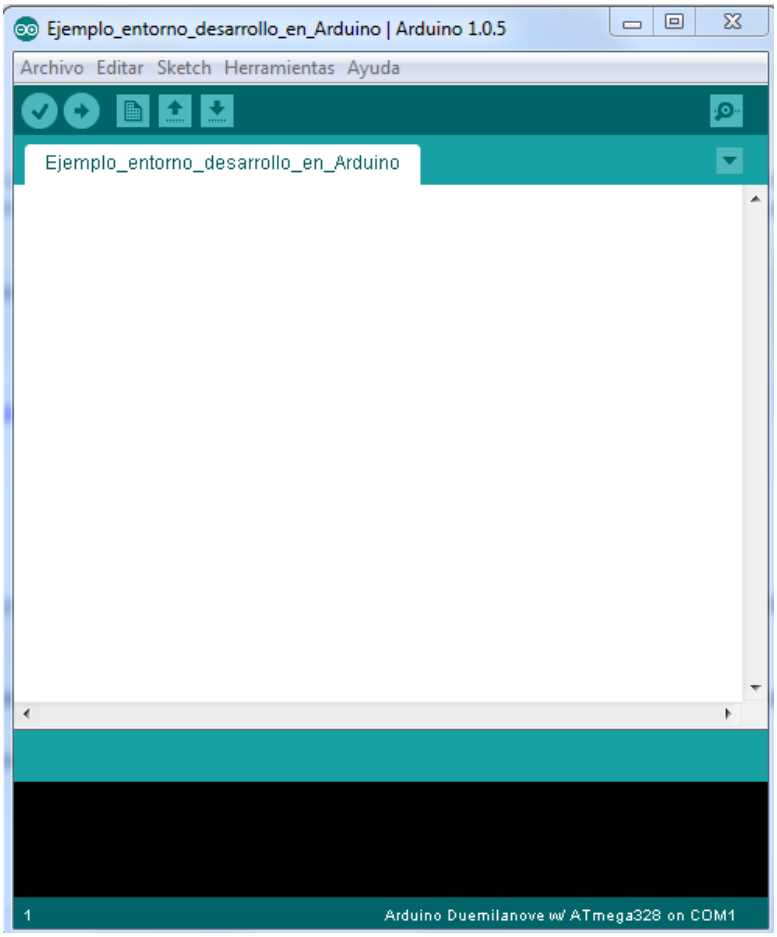


Figura 3.16: Entorno de desarrollo en Arduino

A través de la IDE de Arduino, podemos escribir el código del programa software y crear lo que se conoce por "sketch" (programa). ¿Por qué lo llamamos sketch y no programa? Pues porque el IDE de Arduino viene de Processing, y en este lenguaje de programación enfocado al mundo gráfico, cada código es considerado un boceto, en inglés “sketch”.

El sketch permite la comunicación con la placa Arduino. Estos programas son escritos en el editor de texto, el cual admite las posibilidades de cortar, pegar, buscar y remplazar texto.

En el área de mensajes se muestra, tanto la información mientras se cargan los programas, como los posibles errores que tengamos a la hora de compilar, ya sea por problemas en el código del sketch, por fallo en la detección de nuestro Arduino en el puerto USB, o por cualquier otro problema que sea detectado.

La consola muestra el texto de salida para el entorno de Arduino incluyendo los mensajes de error completos y otras informaciones.

Desde la barra de herramientas tenemos acceso directo a las principales funcionalidades que ofrece el IDE de Arduino, como por ejemplo: verificar el proceso de carga, crear un nuevo sketch, abrir un sketch ya existente, guardar los programas, abrir el Monitor Serial, etc.

A continuación pasamos a describir la utilidad de cada uno de los iconos que aparecen en la pantalla principal del entorno de desarrollo de Arduino:

 "Verificar".- Esta funcionalidad se encarga de verificar el código del sketch en busca de posibles errores. A través del área de mensajes se le notificará al usuario el resultado de dicha verificación. En el caso de que se detecten errores en el código, éstos se detallarán junto con el número de línea en la que han sido detectados. Sólo cuando la comprobación resulta libre de errores podremos proceder a la carga del código en nuestra placa Arduino.

 "Cargar".- Permite compilar el código del sketch y lo carga en Arduino. Cuando la carga a terminado se informa al usuario a través del área de mensajes, y podremos proceder a la apertura del monitor serial.

 "Nuevo".- Para la creación de un nuevo sketch. Abre una nueva hoja de texto donde escribiremos el código correspondiente al sketch.

 "Abrir".- Permite abrir un sketch ya existente que ha sido previamente guardado. También puedes abrir cualquiera de los sketches que trae instalados por defecto el IDE de Arduino.

 "Guardar".- Esta funcionalidad nos permite almacenar el sketch que estemos desarrollando en ese momento. Te permite elegir la ruta en la que quieres guardarlo, y te crea automáticamente una carpeta con el mismo nombre que le des al sketch, guardando éste dentro de la misma.

 "Monitor Serial".- Al pinchar sobre este icono, el entorno de desarrollo de Arduino abre una nueva ventana a través de la cual podemos ver la comunicación establecida por el puerto serie entre la placa Arduino y el PC durante la ejecución del programa. Contiene una barra de escritura mediante la que podemos comunicarnos con Arduino a través de su puerto serie, por ejemplo, para seleccionar distintas opciones que contemple un posible menú creado por el usuario dentro de un código, o para enviar directamente comandos AT a una shield GPRS/GSM que tengamos montada sobre el Arduino. También contempla la opción de

seleccionar el envío de algunos caracteres junto con el texto que introduzcamos en la barra de entrada del mismo, como el carácter de nueva línea, retorno de carro, o los dos. En la figura 3.17 podemos ver la pantalla correspondiente al Monitor Serial y la pestaña desplegable en la que podemos seleccionar las distintas opciones referentes a los caracteres de fin de línea.

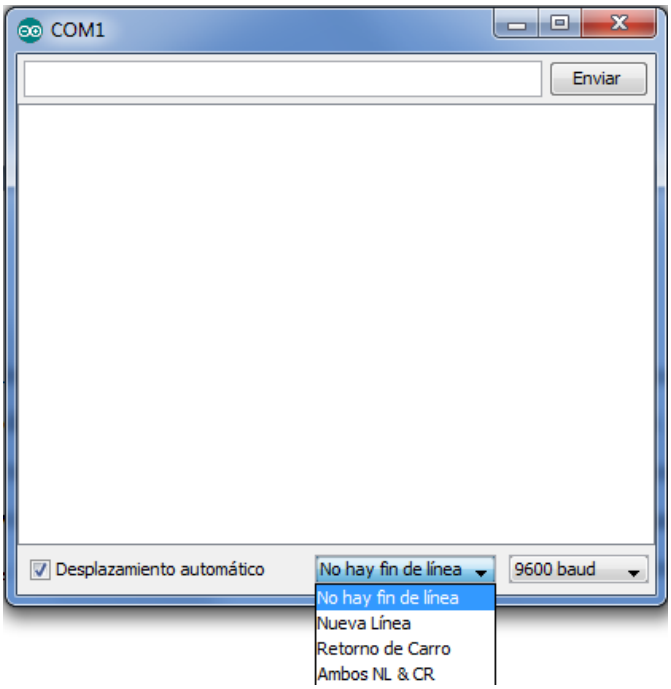


Figura 3.17: Selección de caracteres de fin de línea en la ventana "Monitor Serial"

Dentro del Monitor Serial disponemos de otra pestaña para establecer la tasa de baudios (Baudrate), que marca el número de unidades de señal transmitidas por segundo. Este valor ha de estar sincronizado con el baudrate en el que esté trabajando el Arduino, el cual puede ser establecido en el código del sketch mediante el comando Serial.begin("*valor del baudrate*"), o de no ser así, se establecerá un valor por defecto. Si Monitor Serial y Arduino no están sincronizados con la misma tasa de baudios, la información que aparezca en la ventana será completamente ilegible. En la figura 3.18 aparece desplegada la pestaña para la selección de los distintos valores de baudrate disponibles.

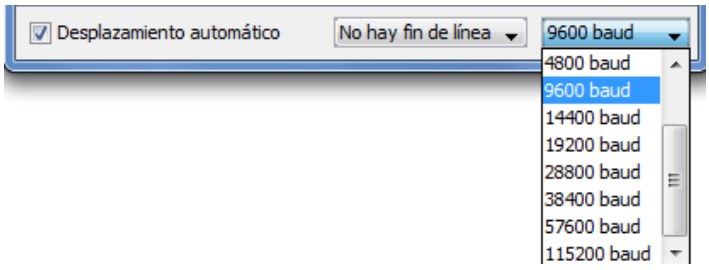


Figura 3.18: Selección del valor de 'baudrate' en la ventana "Monitor Serial

Dentro de los menús, cabe mencionar la existencia de librerías, que pueden proporcionar funcionalidades adicionales para la utilización en sketches, por ejemplo para trabajar con hardware o manipular datos. Para utilizar una librería dentro de un sketch, debemos declararla previamente. Para ello nos iremos al menú "sketch", y seleccionaremos la opción importar librerías. Dentro buscaremos la librería que sea de nuestro interés y la importaremos al sketch, insertando una sentencia de tipo *#include* al comienzo del mismo. Se debe tener en cuenta que al cargar un código que incluya librerías, éstas también se vuelcan en la placa junto con el resto del sketch, incrementando la ocupación del programa y reduciendo el espacio disponible en la memoria de Arduino.

3.3. SHIELDS GPRS/GSM

Si queremos ampliar las funcionalidades de nuestra plataforma Arduino, siempre podemos recurrir a u na gran variedad de shields compatibles prácticamente con cualquiera de sus modelos. De este modo, podemos dotar al dispositivo de funciones adicionales dedicadas específicamente a ofrecer algún tipo de servicio concreto.

Un shield es un módulo de expansión en forma de placa impresa que se puede conectar a la parte superior de la placa Arduino para ampliar sus capacidades, permitiendo además ser apiladas unas encima de otras manteniendo un diseño modular, tal como podemos ver en la Figura 3.19.

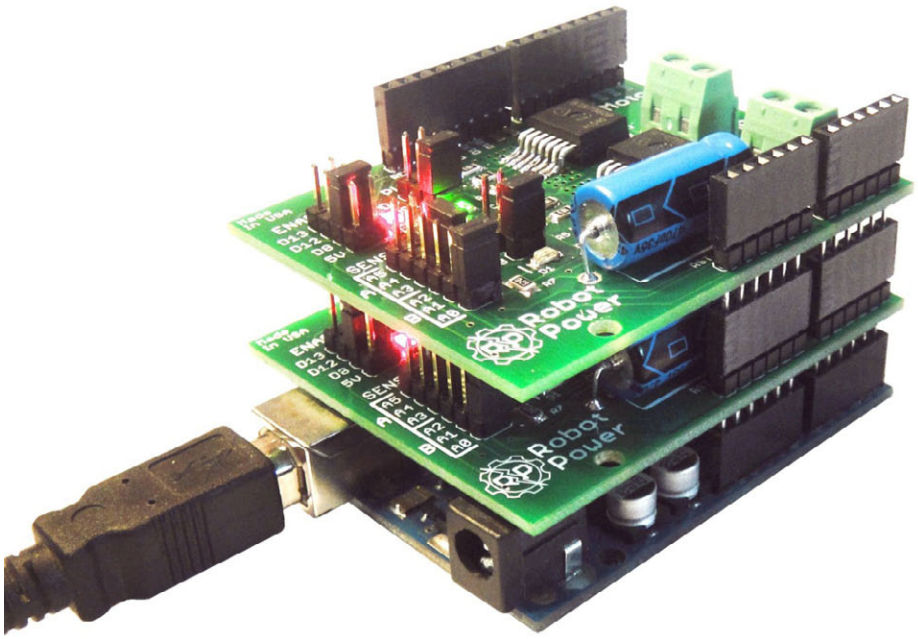


Figura 3.19: Ejemplo de estructura modular con varias shields para Arduino

En nuestro caso, buscamos una shield que nos permita utilizar los sistemas de comunicaciones móviles para poder interactuar a distancia con nuestra plataforma. Navegando por internet podemos encontrar varias shields que han sido diseñadas específicamente para ofrecer servicios a través de los sistemas GSM, GPRS, 3G, o una combinación de los mismos. Además, son perfectamente compatibles con nuestra placa Arduino. [13]

- MÓDULO GPRS QUADBAND PARA ARDUINO:

Esta placa integra un módulo HILO SAGEM que nos permite ofrecer los servicios de un módem GPRS a través de nuestro Arduino. Con esta shield, podemos enviar SMS o hacer llamadas perdidas a otros dispositivos móviles. Es necesaria la conexión de una antena externa para poder establecer las comunicaciones. El precio de esta shield ronda los 86 €.

Figura 3.20: Módulo GPRS Quadband para Arduino

En el esquema reflejado en la figura 3.21 aparece indicada la función de cada uno de los puertos que componen la placa:

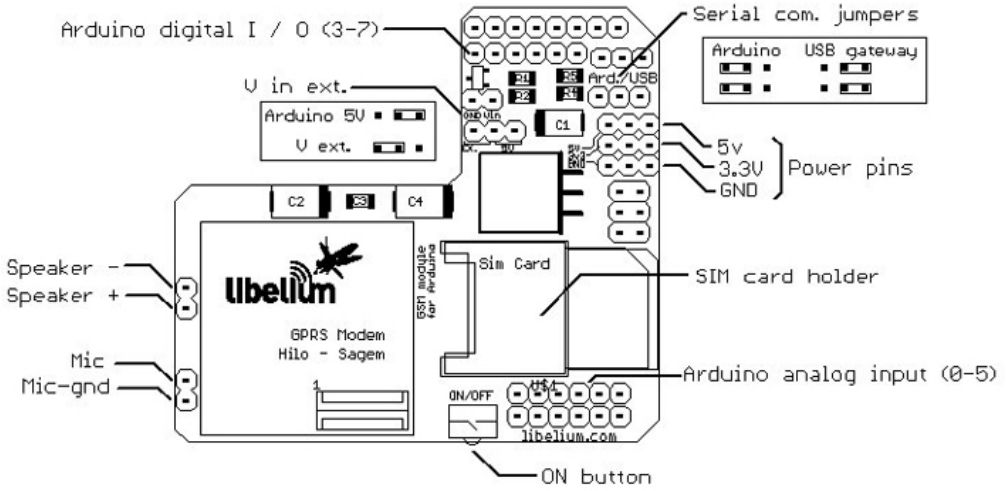


Figura 3.21: Diagrama de puertos y conexiones del módulo GPRS Quadband

En principio se trata de un m ódulo que se adapta perfectamente a lo que estábamos buscando, pero más adelante veremos que existen placas más económicas que ofrecen exactamente las mismas funcionalidades, por lo que finalmente quedó descartada.

- MÓDULO GPRS+GPS QUADBAND PARA ARDUINO/RASPBERRY PI (SIM908)

Este es el último modelo de la shield GPRS para Arduino. Gracias a que cuenta con un módulo SIM908 integrado en la propia placa, ofrece la posibilidad de utilizar la tecnología GPS para posicionamiento en tiempo real, resultando muy útil para aquellas aplicaciones en las que necesitemos conocer la ubicación de nuestro dispositivo. En la figura 3.22 se adjunta una imagen de dicha shield.

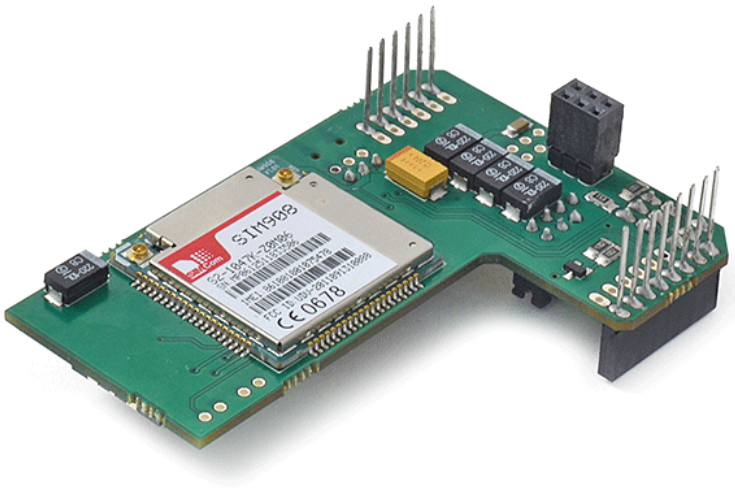


Figura 3.22: Módulo GPRS+GPS Quadband para Arduino y Raspberry PI (SIM 908)

Gracias a este módulo de expansión compatible con la mayoría de las placas Arduino, no sólo podemos transmitir la ubicación del sistema a través de HTTP y almacenarla en un servidor web, sino que también podemos utilizar Google Maps para ver su localización sobre el mapa en todo momento.

Al igual que el resto de placas de este estilo, la antena debe adquirirse por separado, sólo que en este caso necesitaríamos también una segunda antena específica para GPS. El coste de las antenas ronda los 10 €, lo cual no supone una gran inversión comparado con el coste de lo que es la placa en sí, el cual está entorno a los 120 €.

En la figura 3.23 podemos observar con detalle cada uno de los pines, puertos y distintos elementos que conforman esta placa:

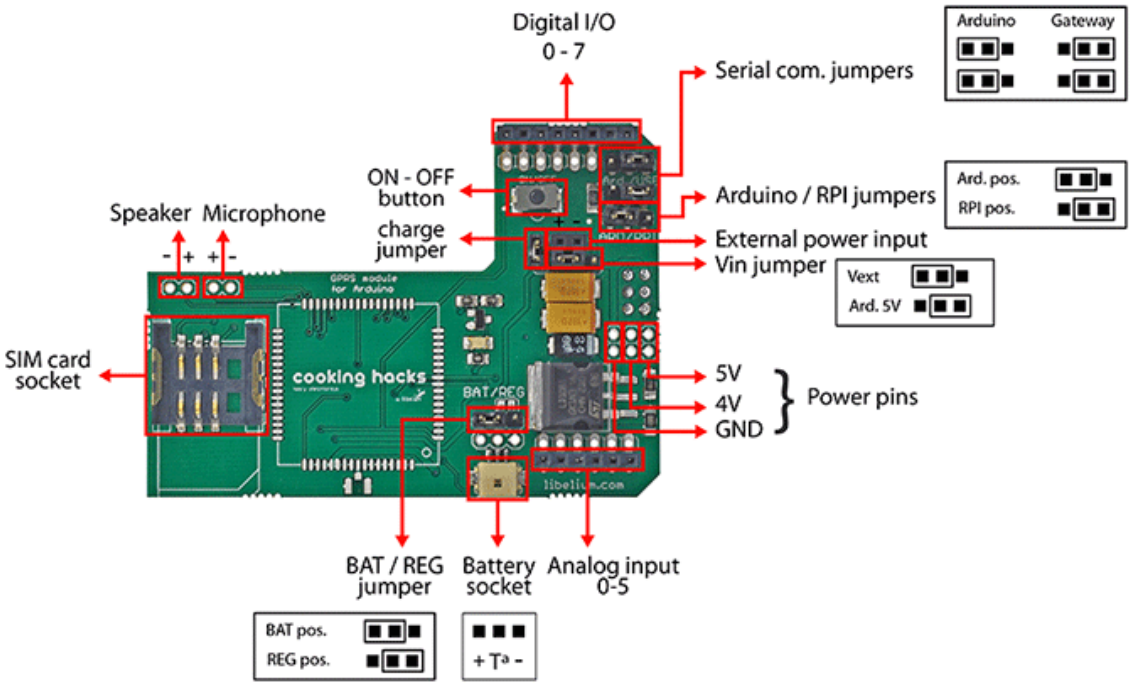


Figura 3.23: Diagrama de conexiones del módulo GPRS+GPS Quadband (SIM908)

En nuestro caso, no tenemos intención de utilizar la tecnología GPS, por tanto no tiene sentido decantarnos por ésta placa. Además, con nuestro planteamiento de reducir los costes en la medida de lo posible, no encajaría escoger esta shield teniendo otros modelos más económicos que ofrecen estrictamente los servicios que necesitamos. Si bien, para aquellos proyectos en los que se necesite ubicar la posición del dispositivo, puede ser un modelo muy interesante, y podremos tenerlo en cuenta a la hora de ofrecer funciones adicionales para nuestro sistema en desarrollo.

- MÓDULO 3G/GPRS+GPS PARA ARDUINO/RASPBERRY PI

Este es el modelo más completo entre todas las shields GPRS disponibles. A parte del sistema GPRS, gracias a su módulo SIM5218, integra también servicios 3G y tecnología GPS. Su precio es bastante elevado, unos 180€, pero es cierto que admite más funcionalidades en comparación con el resto de shields que hemos visto hasta ahora, incluso permite la conexión de una cámara para la toma de imágenes. En la figura 3.24 podemos ver el aspecto que presenta esta shield.

Figura 3.24: Módulo 3G/GPRS+GPS para Arduino/Raspberry PI

A continuación se detalla la estructura de ésta placa:

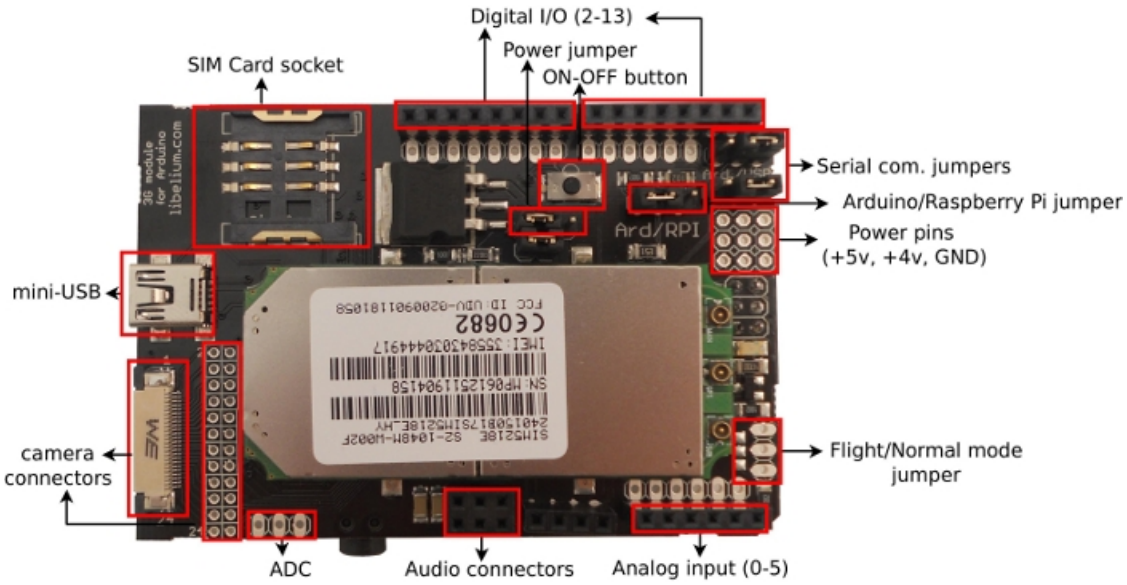


Figura 3.25: Diagrama de puertos y conexiones del módulo 3G/GPRS+GPS

A pesar de que se trata de la placa con mayores prestaciones de entre todas las que hemos podido encontrar, no tiene ningún sentido decantarnos por ella, pues aparte de que su precio está muy por encima del presupuesto inicial fijado para el proyecto (no más de 100 €), incluye tecnologías de las cuales no vamos a hacer ningún uso (GPS y 3G). Desestimamos por tanto, la opción de adquirir esta shield.

- MÓDULO GPRS/GSM QUADBAND PARA ARDUINO (SIM900)

Esta shield puede convertir nuestra placa Arduino en un pl ataforma capaz de ofrecer conectividad GPRS/GSM. Integra un m ódulo SIM900 que nos permite establecer llamadas con otros dispositivos móviles, enviar SMS, incluso la comunicación de datos a través de los protocolos TCP, UDP, HTTP o FTP. Su precio resulta muy interesante, ronda los 75 €. En la figura 3.26 se adjunta una imagen de dicho módulo:

Figura 3.26: Módulo GPRS/GSM Quadband para Arduino (SIM900)

A continuación se adjunta el esquema de diseño correspondiente a dicho módulo:

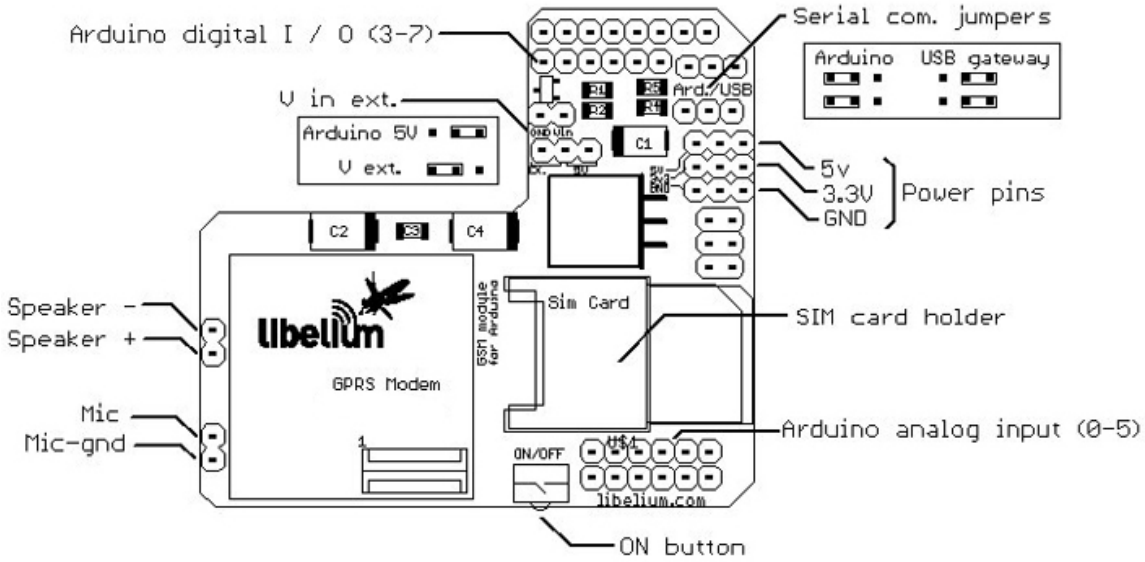


Figura 3.27: Diagrama de conexiones del módulo GPRS/GSM Quadband (SIM900)

No cabe duda de que se trata de uno de los módulos más baratos que podemos encontrar en el mercado de las shields para el uso de sistemas de comunicaciones móviles. No sólo encaja con nuestro presupuesto, sino que nos ofrece todos los servicios que necesitamos para poder llevar a cabo el desarrollo de nuestro dispositivo.

Finalmente, puesto que se trata de la placa con mejor relación precio-prestaciones que hemos podido encontrar, nos decantamos por este modelo para integrarlo en nuestra plataforma.

3.4. SENSORES DE TEMPERATURA

A continuación procedemos a evaluar los principales sensores de temperatura que hemos podido encontrar en el mercado y que son totalmente compatibles con las distintas versiones de la plataforma Arduino.

Básicamente nos decantaremos por un tipo de sensor u otro en función de algunas de sus características (precio, robustez, velocidad de respuesta, etc.), aunque también tendremos en cuenta qué pines serán los que ocupe dicho sensor, de manera que en la medida de lo posible, no interfieran con otra serie de elementos o dispositivos que serán conectados a nuestra placa, como pueden ser el módulo GSM/GPRS, el motor servo, los LEDs, el buzzer, etc.

Principalmente existen tres modelos distintos de sensores de temperatura recomendados para utilizar con Arduino. En concreto, hay un mismo tipo muy compacto disponible en dos modelos, uno que se conecta a pines analógicos, y otro idéntico pero que va conectado a los digitales. El tercer modelo (analógico) consiste en un par de cables con una cabeza metálica, el cual, a priori, parece el más robusto de los tres.

A continuación entramos a ver en detalle cada uno de los modelos, evaluando sus principales características, y todo lo necesario para su montaje sobre la plataforma Arduino.

- SENSOR DE TEMPERATURA DIGITAL “DS18B20”

Comenzamos por el primero de los dos sensores que mantienen el mismo diseño, concretamente en su versión digital, el modelo DS18B20. En la figura 3.28 vemos una imagen del diseño que presenta este tipo de sensor, y podemos hacernos una idea de su reducido tamaño.

Figura 3.28: Sensor de temperatura DS18B20

Como vemos se trata de un elemento muy compacto, compuesto por una pequeña cabeza de plástico y tres patillas metálicas. En apariencia, resulta muy similar al diseño de un transistor.

Para su montaje sobre la placa Arduino, una de sus patillas irá directamente conectada al pin 5V (alimentación); otra debe conectarse a masa (pin GND); y la tercera es la que irá conectada a uno de los pines digitales. Entre la patilla que recibe la alimentación y la que va conectada al pin digital (señal), debe conectarse una resistencia de pull-up de 4,7kΩ. En la figura 3.29 se muestra el esquema de conexión:

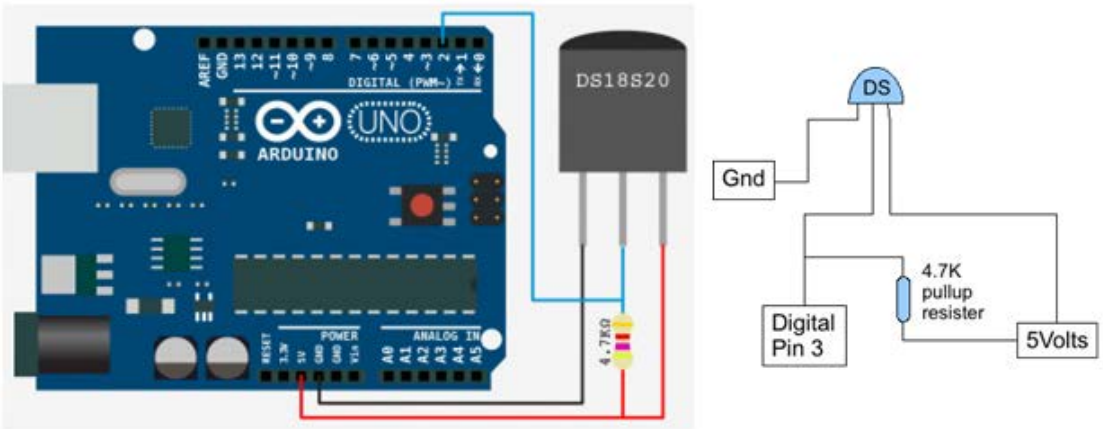


Figura 3.29: Esquema de montaje para la prueba del sensor DS18B20

Su precio, que ronda los 5€, es bastante elevado en comparación con el de los dos modelos analógicos (en torno a 2€), por lo que rápidamente quedó descartado como sensor a utilizar para éste proyecto.

- SENSOR DE TEMPERATURA ANALÓGICO “TMP36”

En cuanto a diseño se refiere, (véase la figura 3.30) podemos decir que el TMP36 es la versión equivalente al modelo que acabamos de presentar.

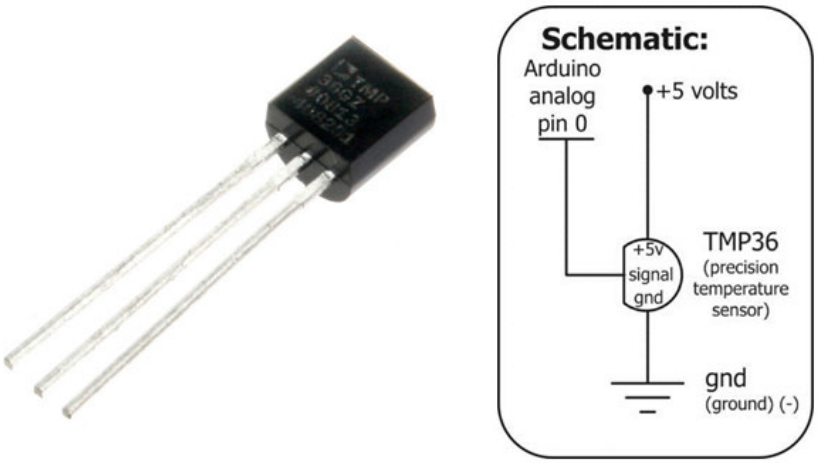


Figura 3.30: Sensor de temperatura TMP36

A diferencia del DS18B20 (modelo digital), el TMP36 necesita ser conectado a uno de los pines analógicos en lugar de a uno digital. En nuestro caso esto no supone ningún problema, pues aun cuando tenemos conectados todos los elementos que precisa nuestro proyecto, seguimos teniendo libres varias entradas/salidas, tanto analógicas como digitales.

El esquema para la conexión del sensor queda representado en la figura 3.31:

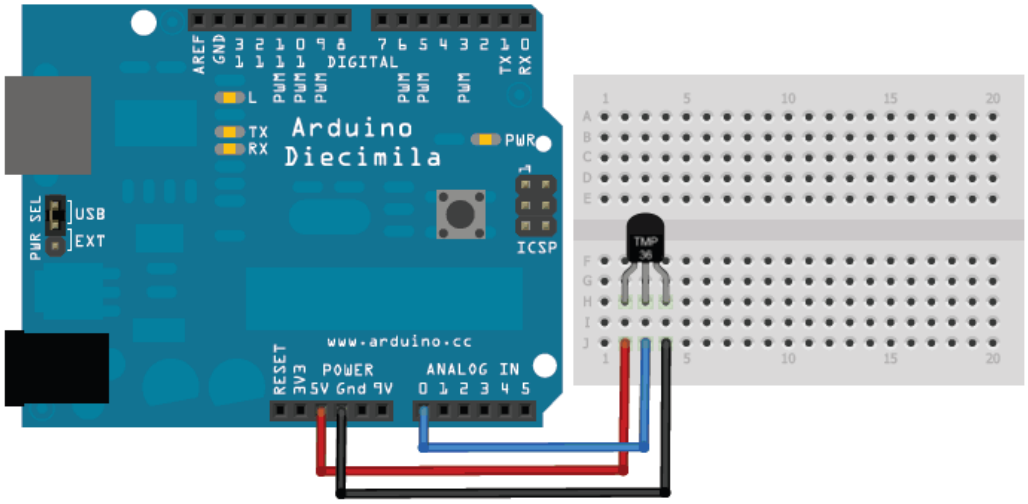


Figura 3.31: Esquema de montaje para la prueba del sensor TMP36

Su precio está por debajo de los 2 €, por lo que resulta un sensor bastante interesante. Tras llevar a c abo las pruebas (reflejadas en el apartado 4, correspondiente al desarrollo experimental) vemos que cumple perfectamente con su función. Estima el valor de temperatura con notable precisión y detecta variaciones en la misma de manera casi instantánea. Sin duda podría ser el tipo de sensor escogido para nuestra plataforma.

- SENSOR DE TEMPERATURA ANALÓGICO “SEN118A2B”

El SEN118A2B es un sensor de temperatura que difiere bastante de los que hemos visto hasta ahora. Su diseño llama la atención por su cabeza de acero, y consta de un cable de dos hilos para su conexión. A priori parece el más robusto de los tres sensores presentados. Además estéticamente hay que darle un punto a su favor, ya que el acabado es metálico y no de pastico como el resto de modelos vistos hasta ahora, tal y como se muestra en la siguiente figura 3.32.

Figura 3.32: Sensor de temperatura SEN118A2B

Para el montaje de este sensor conectaremos directamente uno de los hilos (es indiferente cual) al pin 5V de Arduino, y el otro lo conectaremos a uno de los pines analógicos. Además, entre masa (pin GND de Arduino) y el pin analógico donde se ha conectado dicho hilo, debemos intercalar una resistencia de 10kΩ (valor impuesto por el fabricante). En la figura 3.33 se muestra el esquema de conexión.

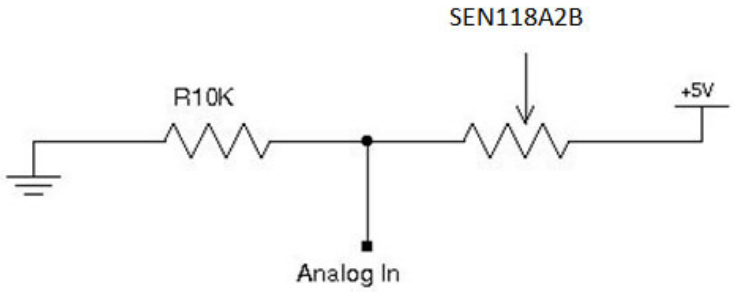


Figura 3.33: Esquema de montaje para la prueba del sensor SEN118A2B

El precio de mercado para este sensor es inferior a los 2 €, prácticamente el mismo que para el modelo TMP36 que veíamos justo antes, por tanto, éste no s erá un f actor que condicione nuestra decisión. Sin embargo, gracias a su diseño, robustez y respuesta casi inmediata frente a cambios en la temperatura ambiente, nos decantamos finalmente por este sensor para integrarlo en nuestro prototipo en desarrollo.

4. DESARROLLO EXPERIMENTAL

En este apartado se exponen paso a paso cada una de las etapas por las que se han ido pasando hasta alcanzar el objetivo final del proyecto, incluyendo: pruebas realizadas, problemas que han ido surgiendo en el desarrollo de las mismas, códigos de programación correspondientes a cada una de las funciones que se han ido probando, y por supuesto, la descripción del sistema que finalmente ha sido diseñado, junto con las pruebas a las que ha sido sometido para evaluar su correcto funcionamiento.

4.1. INSTALANDO EL AMBIENTE DE DESARROLLO DE ARDUINO

Para poder hacer uso de las distintas funciones que ofrece Arduino, es indispensable haber instalado previamente el entorno de desarrollo. A través del mismo, generaremos los sketch con el código del programa que queramos cargar en nuestro dispositivo.

En primer lugar, descargamos la última versión de software disponible en la propia web de Arduino. [14] Puedes descargar el IDE conforme al sistema operativo que tengas, Windows, Mac OS X o Linux. Una vez descargado, ejecutamos el archivo y seguimos los pasos del asistente de instalación. Este paso no debe suponer ningún problema, su instalación es muy sencilla. En Windows, el Arduino es detectado automáticamente, pero en el caso Mac, tendremos que determinar manualmente el Puerto Serie en el que tenemos conectada nuestra placa. Tal como se muestra en la figura 4.1, desde el menú "Tool" podemos seleccionar el puerto correspondiente, que será el que comience por */dev/cu.usbmodem*. Éste es el nombre con el que MAC hace referencia al Arduino. Eso sí, no olvidemos tenerlo conectado al puerto USB, de no ser así, no aparecerá ésta descripción en dicho puerto y no podremos identificarlo.

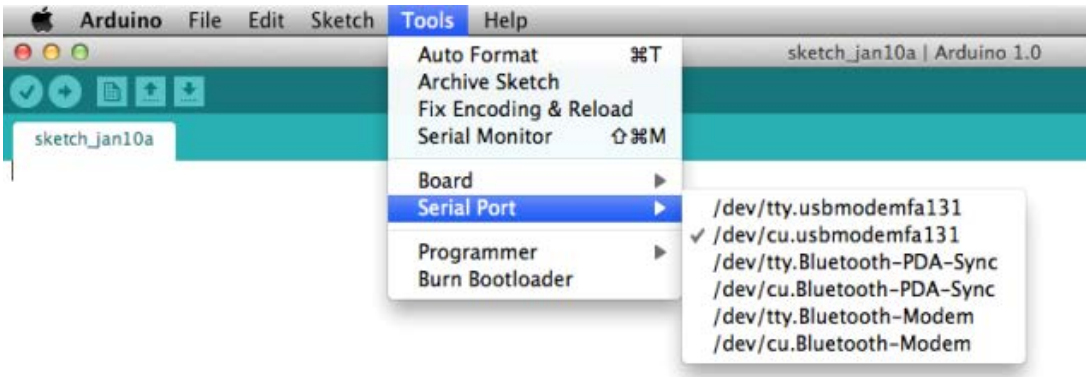


Figura 4.1: Selección del puerto serie en el IDE de Arduino para MAC

Una vez completada la instalación e identificado el puerto en el que tenemos conectado nuestro Arduino, llega el turno de seleccionar el modelo concreto de nuestra placa. Para ello accedemos al menú “Herramientas”, dentro del mismo colocamos el cursor sobre la pestaña “Board”, y a continuación clicamos sobre la versión correspondiente a nuestro modelo, en nuestro caso, “Arduino Uno”. En la figura 4.2 podemos ver el procedimiento.

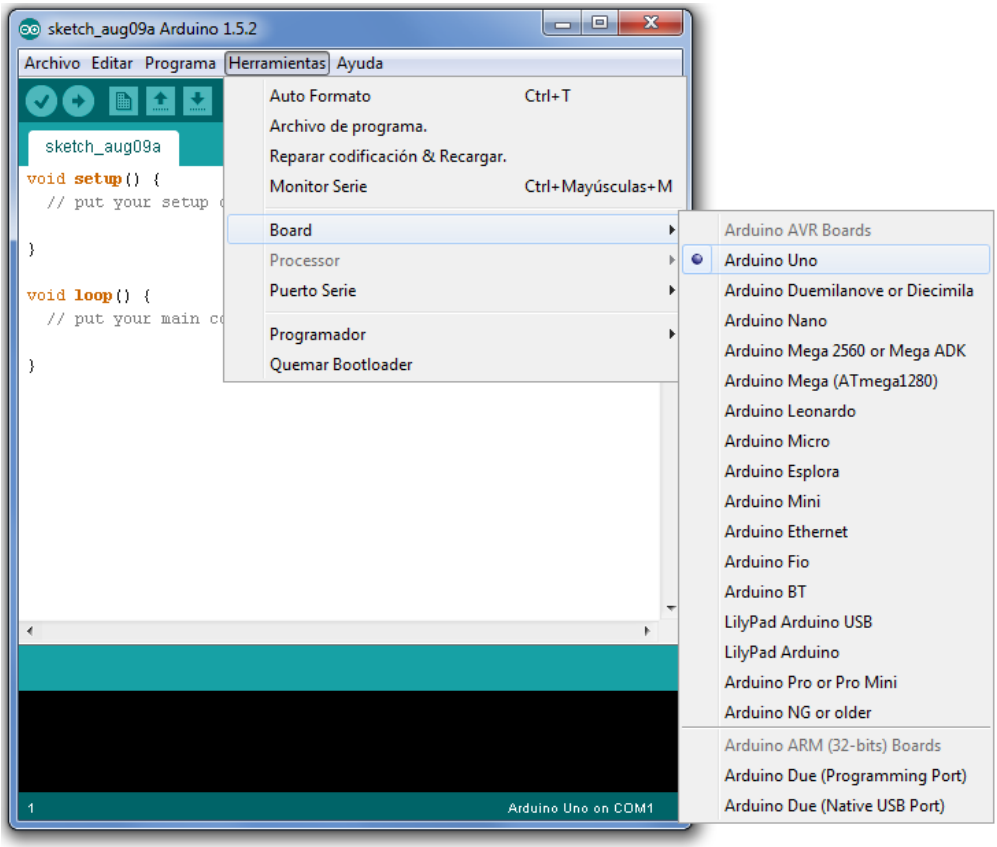


Figura 4.2: Ventana para la selección del modelo de placa de Arduino

Llegados a este punto, ya tenemos instalado y configurado el IDE de Arduino. Para comenzar a trabajar con nuestra placa, sólo tenemos que generar un nuevo sketch y comenzar a escribir el código de programación con las instrucciones que queremos que ejecute nuestro Arduino.

4.2. COMENZANDO CON ARDUINO. HOLA MUNDO

A partir de este momento comenzamos a adentrarnos de verdad en la parte experimental del proyecto. Como es lógico, hasta llegar a construir un dispositivo de cierta complejidad debemos pasar previamente por una serie de etapas más sencillas hasta lograr alcanzar el objetivo final.

Para irnos familiarizando con el entorno de desarrollo de Arduino e ir cogiendo la mecánica para la carga y ejecución de programas, comenzamos por el sketch (programa) más sencillo, el llamado "Hola Mundo" de Arduino. [15] Consiste en hacer parpadear un LED de acuerdo a unos intervalos de tiempo predefinidos por el usuario en el código del sketch.

Las únicas herramientas con las que debemos contar para seguir este primer ejercicio son: una placa Arduino y un LED.

En nuestro caso, contamos con el modelo Arduino UNO. Esta versión está diseñada para que hacer parpadear un LED sea muy sencillo, de hecho ya incorpora un LED integrado en la propia placa, pero también permite conectar uno externamente utilizando el pin digital 13. Para conectar un LED a otro pin digital que no sea éste, se recomienda intercalar una resistencia externa de 1kΩ.

Como podemos ver en la figura 4.3, el montaje no supone ninguna complicación, simplemente debemos identificar el ánodo y el cátodo del LED, y conectarlos directamente en el pin 13 y GND respectivamente.

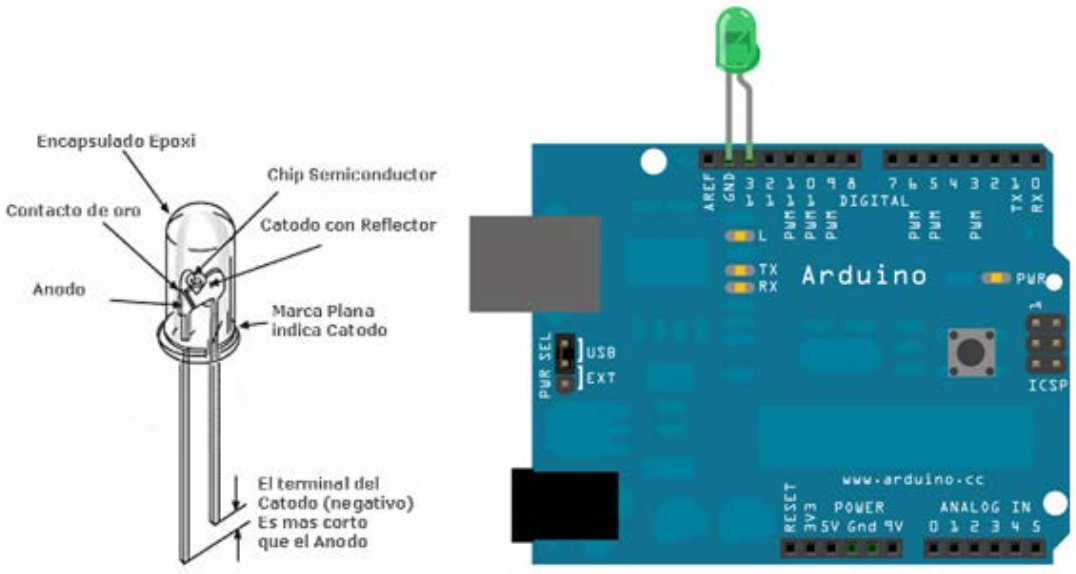


Figura 4.3: Esquema de montaje para hacer parpadear un LED con Arduino

A continuación, debemos generar el código con las instrucciones que queremos que lleve a cabo el Arduino. Para este ejemplo ni siquiera es necesario que escribamos el código, pues podemos importarlo directamente desde los ejemplos que incluye por defecto el IDE de Arduino. Basta con acceder al menú "Archivo", seleccionar "Ejemplos", luego "01.Basics", y por último "Blink" (en inglés, *parpadear*). La siguiente figura 4.4 refleja los pasos a seguir sobre la pantalla principal del entorno de desarrollo de Arduino.

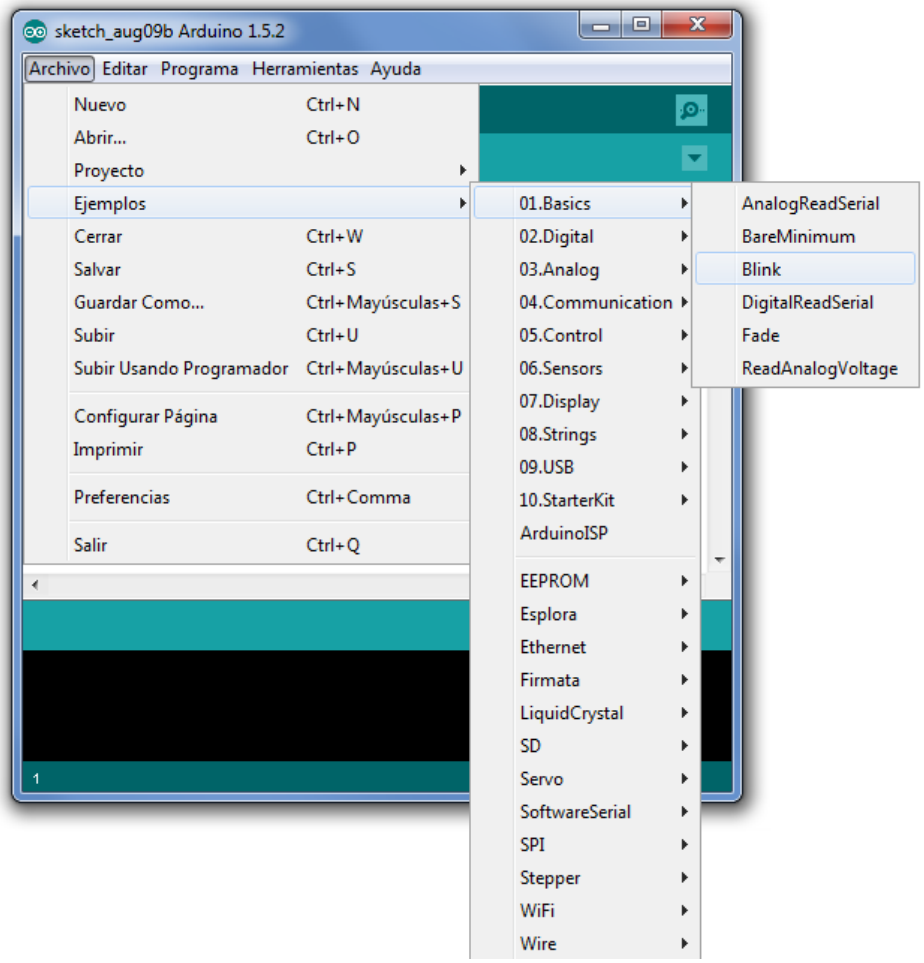
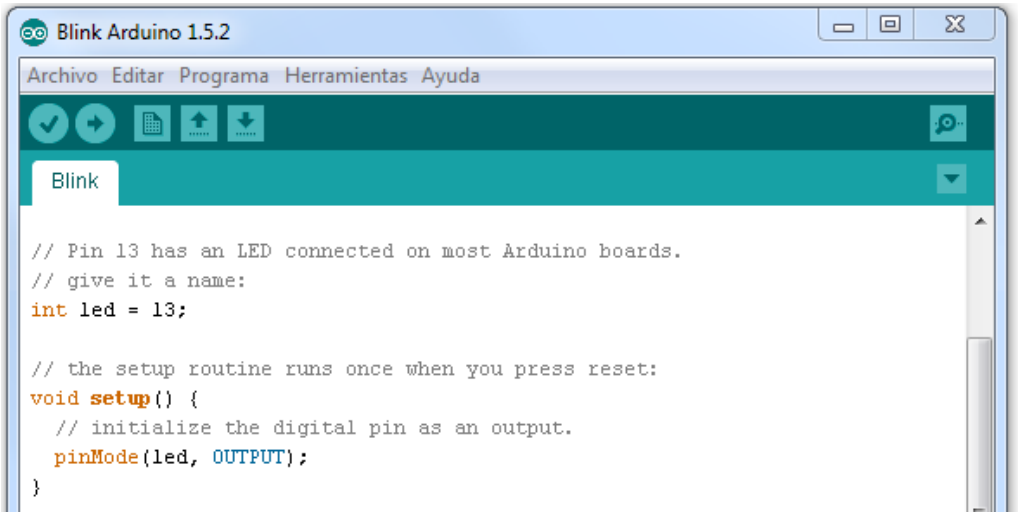


Figura 4.4: Selección de códigos de ejemplo incluidos en el IDE de Arduino

Tras seguir estas instrucciones se abrirá una nueva pestaña que contiene el sketch correspondiente al programa para hacer parpadear un LED. En la figura 4.5 podemos ver que el código es bastante sencillo:



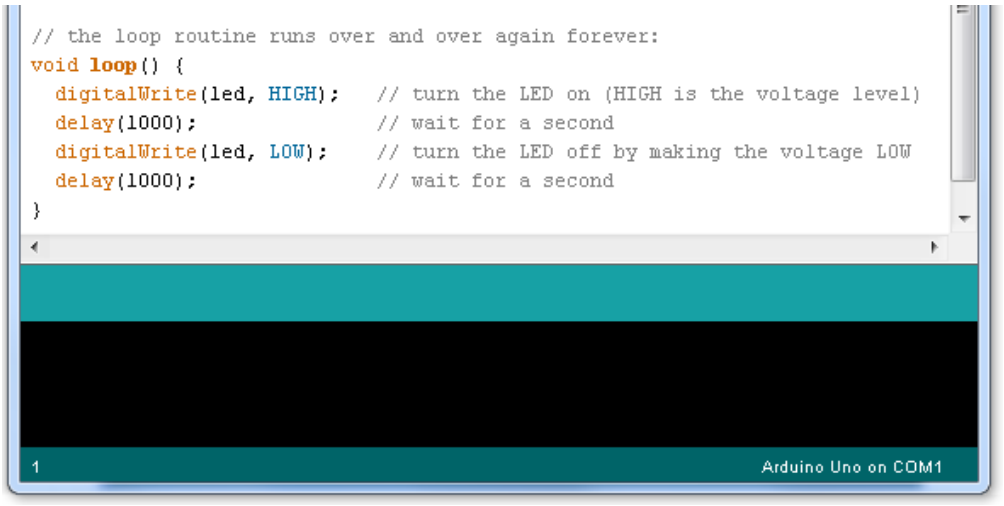


Figura 4.5: Código correspondiente al sketch Blink

Una característica aplicable a todos los sketch, es que siempre deben estar incluidas las funciones "void setup()" y "void loop()", sin ellas cuando compilemos el código siempre nos dará un error, y no podremos proceder con la carga en Arduino.

Una vez que tenemos escrito el código correspondiente al programa, le damos a  "Verificar", y dará paso a la compilación del sketch en busca de posibles errores.

Si todo es correcto, procederemos con la carga del programa, clicando sobre la opción  "Cargar". (Nota: No olvidemos tener conectado nuestro Arduino al puerto USB)

Tal y como se muestra en la figura 4.6, durante la carga, en el área de mensajes aparecerá la palabra "Subiendo", a medida que avanza la barra de progreso. Cuando ésta llegue al final, ya tendremos cargado nuestro programa, y el LED debería comenzar a parpadear.

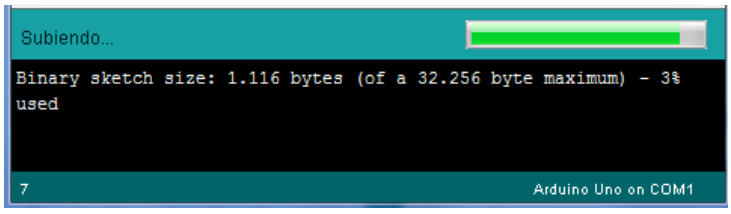


Figura 4.6: Proceso de carga del sketch en Arduino

Si queremos jugar un poco con éste código, podemos variar de manera muy simple la velocidad de parpadeo del LED. Sólo tenemos que modificar los valores de retardo (delay) tras las instrucciones de encendido y apagado. Se debe tener en cuenta que en la instrucción "delay", el valor de retardo a de indicarse en milisegundos. Por ejemplo; para un retardo de medio segundo, la declaración sería: "*delay(500);*".

4.3. EVALUACIÓN DE LOS DISTINTOS SENSORES DE TEMPERATURA

A continuación procedemos a evaluar el comportamiento de los distintos sensores de temperatura que presentamos anteriormente en el apartado “3.4. Sensores de Temperatura”.

Aunque se va a realizar el proyecto sobre un sensor de temperatura, es un ejemplo que podría aplicarse a la mayor parte de los sensores existentes, como sensores de luminosidad, humedad, presencia, etc., ofreciendo un amplio abanico de posibilidades de cara a la expansión de nuestro sistema o para su adaptación a otro tipo aplicaciones relacionadas con el sector IOT (Internet of Things).

Entramos en la evaluación de los sensores de temperatura. Recordamos que el sensor digital (DS18B20) quedó descartado desde un principio por su alto precio, por lo que únicamente adquirimos los sensores correspondientes a los modelos analógicos (TMP36 y SEN118A2B), los cuales someteremos a algunas pruebas con el fin de apreciar posibles diferencias en el rendimiento de ambos que justifiquen la elección de un modelo u otro.

- SENSOR DE TEMPERATURA ANALÓGICO “TMP36”

Comenzamos estudiando el modelo más sencillo y compacto, el TMP36. Lo primero que debemos hacer es llevar a cabo el esquema de montaje correspondiente a este modelo de sensor (véase el apartado “3.4. Sensores de Temperatura”). El resultado de la conexión en nuestra placa Arduino queda reflejado en la siguiente figura 4.7:

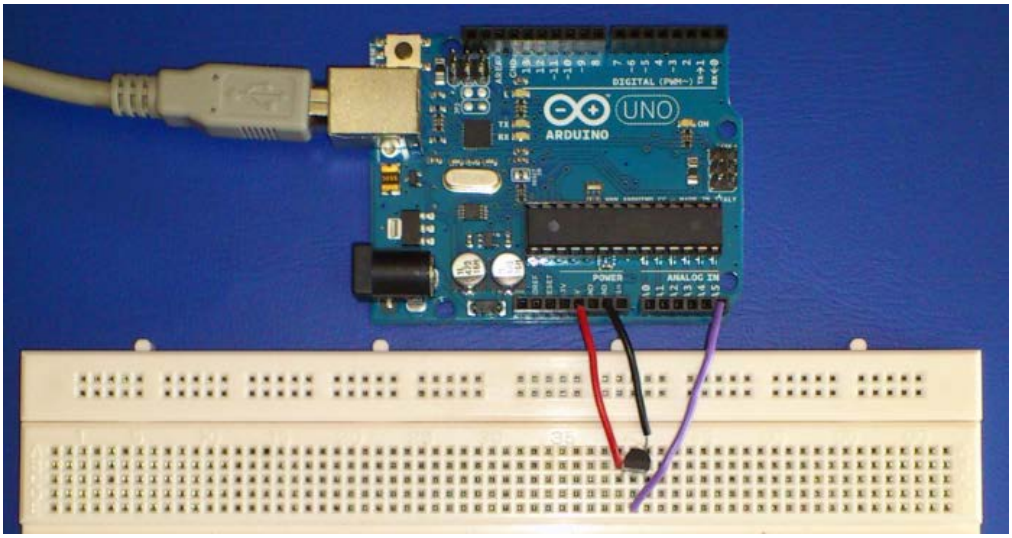


Figura 4.7: Montaje del sensor de temperatura TMP36 sobre la plataforma Arduino UNO

A continuación pasamos a generar un código que calcule el valor de temperatura y nos lo envíe constantemente a través del puerto serie de Arduino.

En este caso, el propio fabricante nos facilita las fórmulas para la conversión del valor de tensión a grados Celsius o Fahrenheit, las cuales podemos encontrar en el datasheet del dispositivo. [16]

Una vez que tenemos los algoritmos de conversión, sólo tenemos que escribir un sencillo programa que se encargue de enviar los datos a través del puerto serie de Arduino. No obstante, también podemos encontrar por internet algunos ejemplos muy simples a modo de tutorial, de donde podemos coger el código directamente. [17]

A continuación se adjunta el código correspondiente a nuestro sketch:

```
int temp = 5;  //Pin analógico A5

//Variables para ir comprobando maximos y minimos
float maxC = 0, minC = 100, maxF = 0, minF = 500, maxV = 0, minV = 5;

void setup(){
  Serial.begin(9600);
}

void loop(){
  float voltaje;
  float gradosC;
  float gradosF;

  voltaje = analogRead(5) * 0.004882814;

 //Con esta operación lo que hacemos es convertir el valor que nos devuelve el
  analogRead(5) que   va a estar comprendido entre 0 y 1023 a un valor comprendido
  entre los 0.0 y los 5.0 voltios

  gradosC = (voltaje - 0.5) * 100.0;
  //Gracias a esta fórmula que viene en el datasheet del sensor podemos convertir el valor
  del voltaje a grados centigrados

  gradosF = ((voltaje - 0.5) * 100.0) * (9.0/5.0) + 32.0;
  //Con esta otra fórmula convertimos el valor del voltaje a grados Fahrenheit
  Serial.print("Temperatura actual:\n");
  Serial.print("Celsius: ");
  Serial.print(gradosC);
  Serial.print("\tFahrenheit: ");
  Serial.print(gradosF);

  //Comprobacion de maximos y minimos de humedad y temperatura
  if (maxC < gradosC)
    maxC = gradosC;
```

```
  if (gradosC < minC)
    minC = gradosC;

  if (maxF < gradosF)
    maxF = gradosF;

  if (gradosF < minF)
    minF = gradosF;

  if (maxV < voltaje)
    maxV = voltaje;

  if (voltaje < minV)
    minV = voltaje;

  Serial.print("\nMaxima temperatura registrada:\n");
  Serial.print("Celsius: ");
  Serial.print(maxC);
  Serial.print("\tFahrenheit: ");
  Serial.print(maxF);

  Serial.print("\nMinima temperatura registrada:\n");
  Serial.print("Celsius: ");
  Serial.print(minC);
  Serial.print("\tFahrenheit: ");
  Serial.print(minF);
  Serial.print("\n\n");

  delay(5000);  //esperamos 5 segundos para una nueva lectura
}
```

Una vez escrito dicho código, pulsamos sobre el icono  "Verificar" para que lo analice en busca de posibles errores. Si todo es correcto, en el área de mensajes aparecerá la palabra 'Compilado', tal y como se muestra en la figura 4.8.

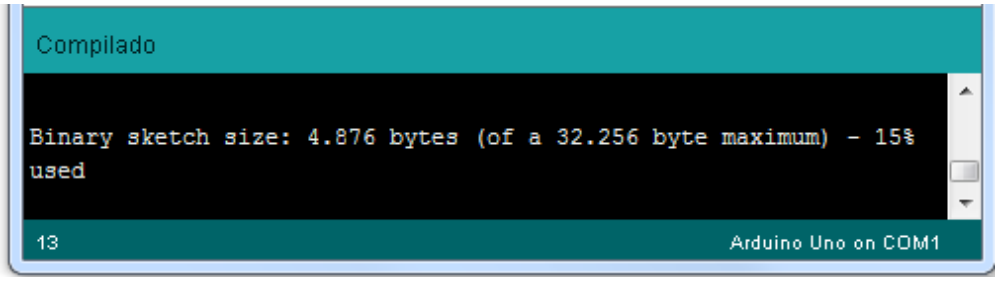


Figura 4.8: Resultado de la compilación del sketch

Lo siguiente será enviar el programa a la memoria Flash de Arduino. Para ello clicamos sobre el icono  "Subir" y esperamos a que finalice la carga.

Una vez cargado el sketch, abrimos el "Monitor Serial" 

y observamos el comportamiento del programa.

Como vemos en la figura 4.9, el programa nos muestra cada 5 segundos la temperatura actual junto con el valor máximo y mínimo registrado hasta ese momento.

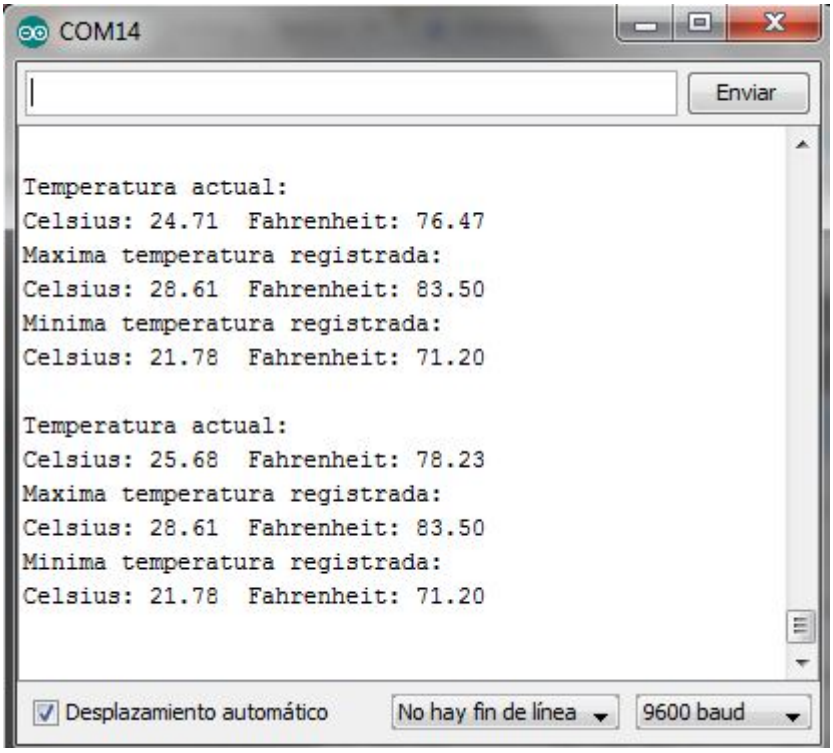


Figura 4.9: Monitor Serial durante la ejecución del programa de prueba del sensor TMP36

Para comprobar la respuesta del sensor frente a cambios de temperatura, modificamos una sola línea de código ("*delay(5000);"*), reduciendo el retardo entre lectura y lectura a 0,5 segundos (nueva instrucción: "*delay(500);"*).

Tras esto, volvemos a ejecutar el programa y acercamos levemente un mechero para aumentar la temperatura ambiente. Enseguida observamos los cambios en el valor de temperatura registrado, y podemos afirmar que la velocidad de respuesta del sensor es inferior a medio segundo.

Este retardo lo podemos considerar despreciable, pues nuestra aplicación no requiere de una consulta instantánea de la temperatura, sino más bien algo periódico, donde no se verá reflejado en ningún momento dicho retardo.

- SENSOR DE TEMPERATURA ANALÓGICO “SEN118A2B”

En esta ocasión procedemos con el análisis del sensor SEN118A2B. En primer lugar conectamos los elementos necesarios de acuerdo al esquema de montaje que adjuntamos previamente en el apartado “3.4. Sensores de Temperatura”. En la siguiente figura 4.10 podemos ver una imagen con el resultado de dicho montaje.

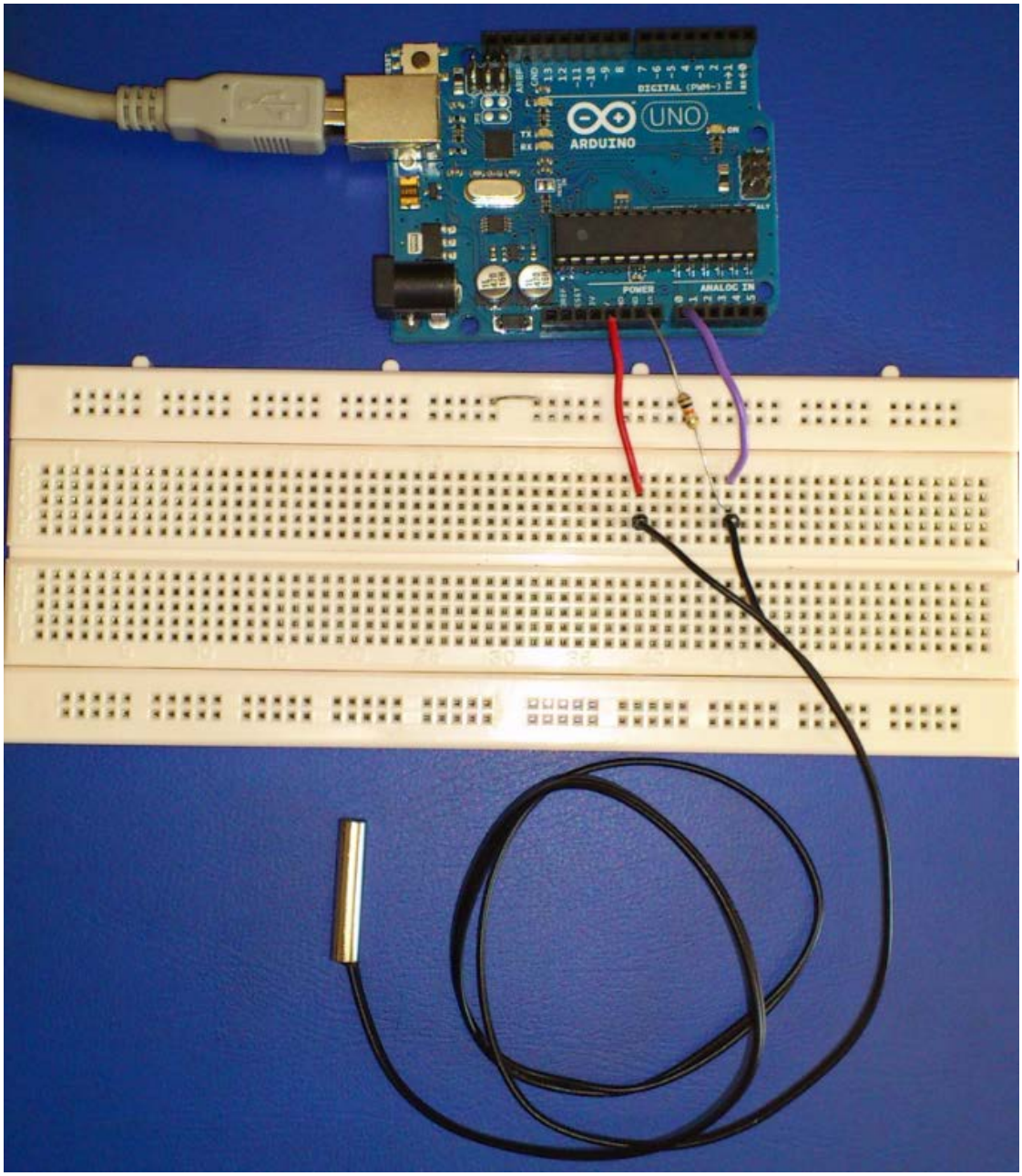


Figura 4.10: Montaje del sensor de temperatura SEN118A2B sobre la plataforma Arduino

A continuación buscamos el algoritmo correspondiente para el cálculo de la temperatura en función del valor obtenido por el sensor (RawADC) y del valor de la resistencia utilizada. En nuestro caso vamos a utilizar una resistencia de 10kΩ, que es la recomendada por el propio fabricante. La fórmula que debemos aplicar es la siguiente:

Temperatura (Kelvin) = 1 / { A + B*[ln(Temp)] + C*[ln(Temp)]*3 };

Dónde: A = 0.001129148 B = 0.000234125 C = 8.76741e-08

Temp = log { (1024*R / RawADC) – R }

Con esta información ya podemos pasar a generar un pequeño programa que obtenga el valor de temperatura y la muestre por pantalla a través del puerto serie de Arduino. El código es el siguiente:

```
#include <math.h>

#define ThermistorPIN 0  // Pin analogico 0

float vcc = 4.91;  // valor medido en el Pin 5V de Arduino
float pad = 9850;  // valor medido de la Resistencia de 10K
float thermr = 10000; // valor nominal de la Resistencia de 10K

float Thermistor(int RawADC) {
  long Resistance;
  float Temp;
  Resistance=((1024 * pad / RawADC) - pad);
  Temp = log(Resistance);
  Temp = 1 / (0.001129148 + (0.000234125 * Temp) +
  (0.0000000876741 * Temp * Temp * Temp));
  Temp = Temp - 273.15;  // pasamos de Kelvin a Celsius
  return Temp;
}

void setup() {
  Serial.begin(9200);
}

void loop() {
  float temp;
  float Celsius;

  celsius = Thermistor(analogRead(ThermistorPIN));
  Serial.print("Temperatura registrada:\n");
  Serial.print("Celsius: ");
  Serial.print(celsius,1);
```

```
    temp = celsius + 273.15;  // convertimos a  Kelvin
    Serial.print("\tKelvin: ");
    Serial.print(temp,1);
    temp = (celsius * 9.0)/ 5.0 + 32.0;  // convertimos a  Fahrenheit
    Serial.print("\tFahrenheit: ");
    Serial.print(temp,1);
    Serial.print("\n\n");
    delay(5000);    //espera entre lectura y lectura de 5 seg
}
```

De igual modo que hicimos con el otro sensor, compilamos y cargamos el código en nuestra placa. Si todo es correcto, podremos abrir el Monitor Serial y ver cómo el programa comienza a ejecutarse, mostrando por pantalla el valor de temperatura, tal y como refleja la figura 4.11.

Figura 4.11: Monitor Serial durante la prueba del sensor SEN118A2B

Inducimos cambios en la temperatura para ver el tiempo de respuesta del sensor y verificar que no siempre muestra el mismo valor.

Observamos que la respuesta es inmediata, y podemos apreciar que este sensor es algo más sensible frente a cambios de temperatura de lo que lo era el otro modelo analógico (TMP36). Puesto que éste sensor (SEN118A2B) ofrece un mejor rendimiento que el anterior y la diferencia en coste es insignificante, elegimos este modelo como dispositivo a utilizar en nuestra plataforma.

4.4. ACCIONAR UN SERVOMOTOR

Tal como venimos comentando, la intención del proyecto no es sólo la de controlar sensores, sino también actuadores. En este caso, el accionamiento de un motor servo nos servirá como ejemplo para demostrar que podemos accionar prácticamente cualquier otro dispositivo de manera similar, como válvulas para el corte de suministros de agua o gas, sistemas de control de persianas, interruptores eléctricos, etc.

Existen distintos servomotores compatibles con Arduino: analógicos, digitales, de giro parcial, de rotación continua, etc.

Nuestra idea es bastante simple. Consiste en accionar un pequeño motor servo de modo que éste, a su vez, haga girar la rueda de un termostato con fin de regular la temperatura de una vivienda. Por lo tanto, con un servomotor de medio giro (180º) será suficiente para llevar a cabo dicha función.

Los modelos analógicos requieren de un potenciómetro externo para poder regular el ángulo de giro del servo motor, lo que complicaría bastante más las cosas. Sin embargo, los modelos digitales son muy sencillos de programar, basta con indicarle los grados que queremos hacer rotar el servo para que lleve a cabo dicho giro. Por este motivo, y dado que el precio es incluso menor en el caso de los digitales (unos 4€), nos inclinamos por éstos para el diseño de nuestro prototipo.

El modelo más utilizado para su integración con Arduino es el que se muestra en la siguiente figura 4.12, el "Micro Servo 9G", de 1,6Kg de torque y 180º de rotación.

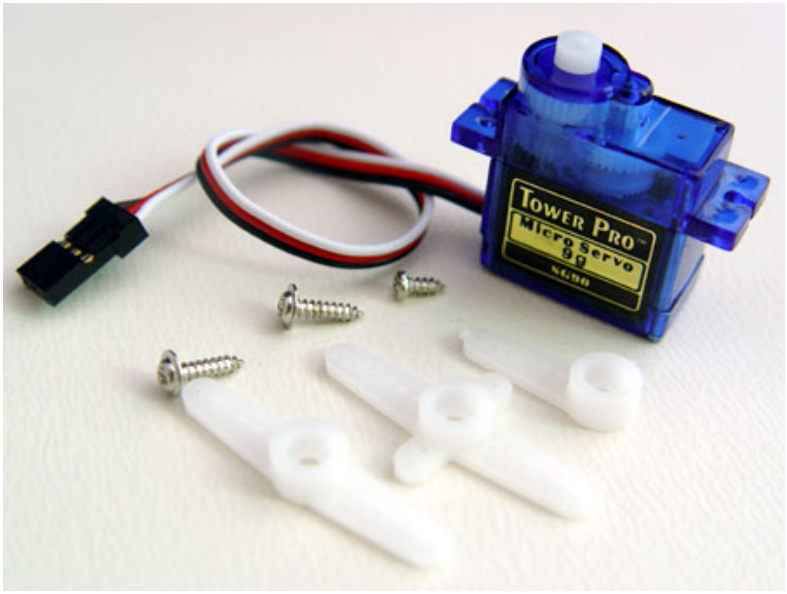


Figura 4.12: Micro Servo 9G

Este servomotor digital (Micro Servo 9G) ha sido el modelo adquirido para ser integrado en nuestra plataforma. Consta de un conector compuesto por tres cables que se distinguen fácilmente por sus colores (rojo, negro y amarillo).

En la siguiente figura podemos observar su correspondiente esquema de montaje:

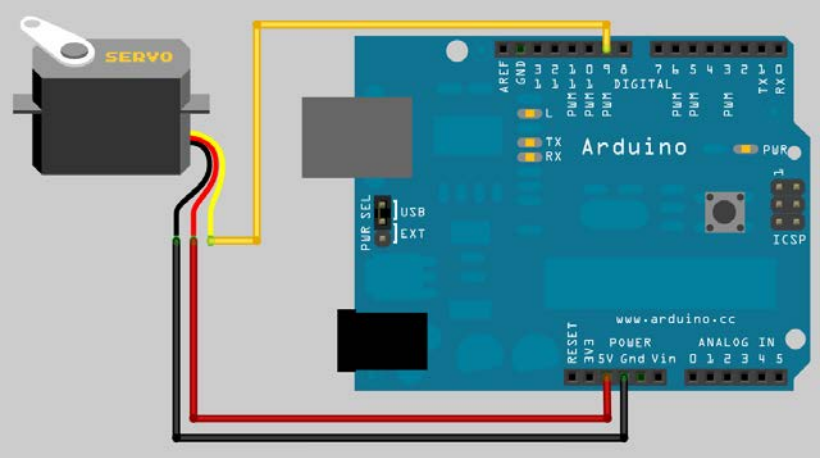


Figura 4.13: Esquema de montaje de motor servo

Como vemos en la figura 4.13, el cable rojo corresponde con la alimentación, e irá conectado al pin 5V de Arduino. El negro corresponde a la toma de tierra, y lo conectaremos directamente al pin GND. Y por último, el amarillo, que irá conectado a uno de los pines digitales de nuestra placa y será el que reciba la señal de control para el accionamiento del motor. Siguiendo estos pasos, procedemos con el montaje sobre nuestra placa Arduino, tal como se muestra en la figura 4.14:

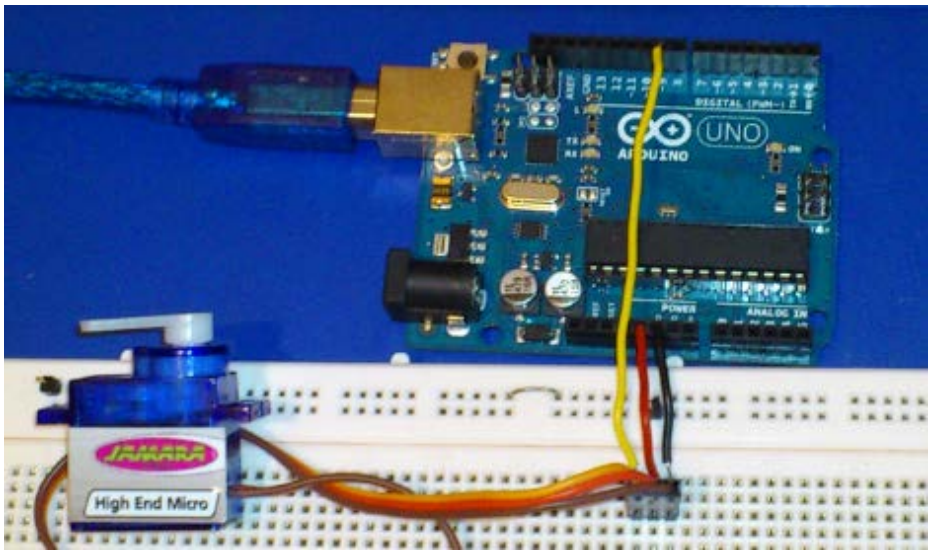


Figura 4.14: Montaje de motor servo sobre la plataforma Arduino UNO

Una vez que lo tenemos montado, conectamos nuestra plataforma al puerto USB del PC. El siguiente paso será generar el sketch con un código muy sencillo a modo de ejemplo para comprobar únicamente el correcto funcionamiento del servomotor.

El IDE de Arduino contiene una librería (Servo.h) que nos facilita el trabajo cuando utilizamos este tipo de dispositivos. Lo primero que debemos hacer es incluir dicha librería. Para ello, nos vamos al menú "Sketch", seleccionamos "Importar librería", y a continuación clicamos sobre la librería "Servo" (véase la siguiente figura 4.15).

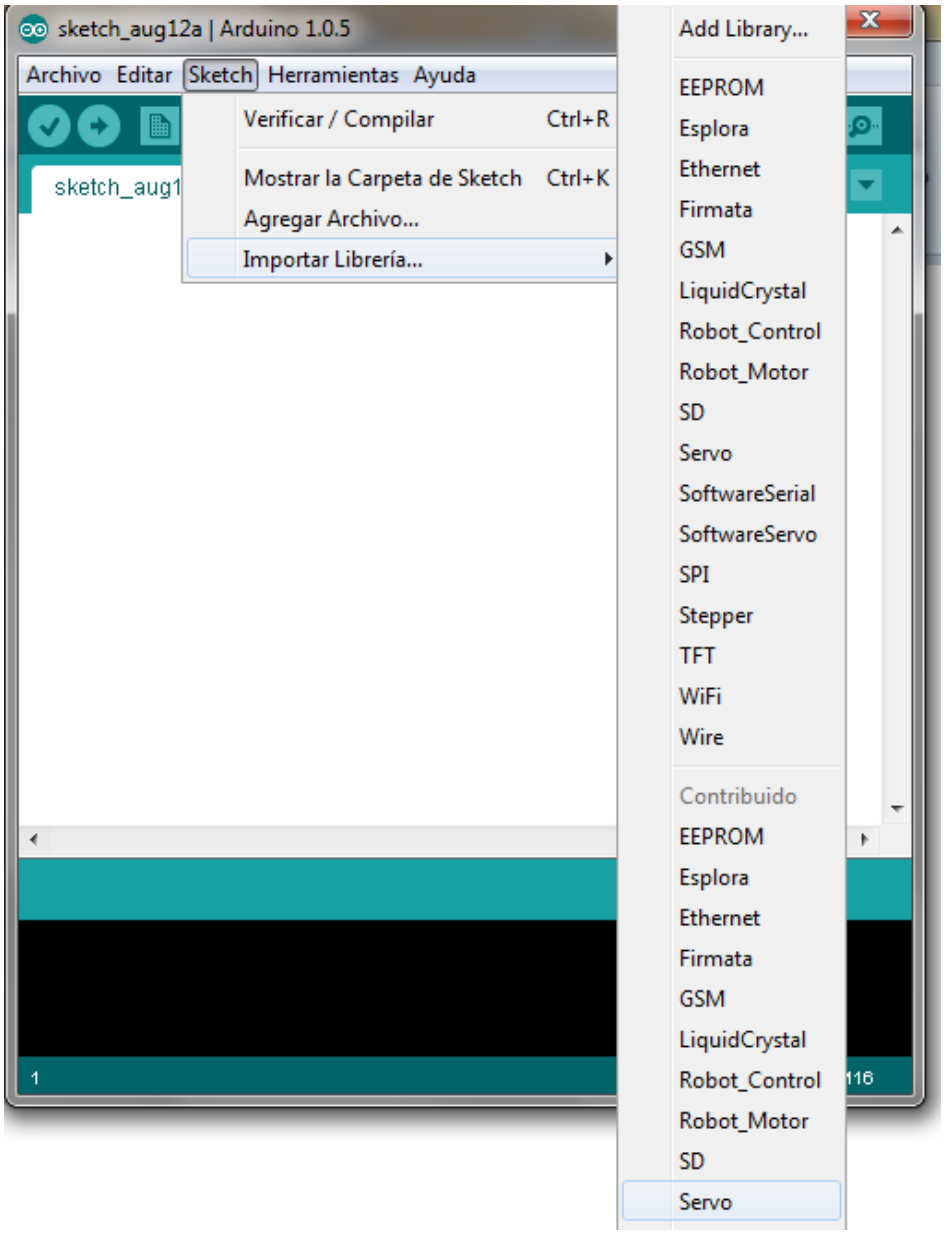


Figura 4.15: Importación de librerías en el IDE de Arduino

En la figura 4.16 se adjunta el resto del código, tal como podemos ver a continuación:

Figura 4.16: Código correspondiente al sketch de prueba de un motor servo

Como siempre, compilamos, y acto seguido cargamos el código en nuestro Arduino. Si todo ha ido bien y no se han detectado errores, el servo comenzará a cambiar de una posición a otra, pasando de 0º a 180º y viceversa en intervalos de un segundo.

Tras esta pequeña prueba podemos afirmar que nuestro servo funciona correctamente. Ahora solo falta integrar esta funcionalidad en el resto del programa en desarrollo, de manera que podamos actuar sobre la rueda de un termostato seleccionando un determinado valor de temperatura.

4.5. SISTEMA DE ALARMA CON UN BUZZER Y UN LED

En esta ocasión vamos a probar un pequeño sistema de alarma compuesto por un buzzer (bocina) y un LED.

La idea consiste en sincronizar el parpadeo de un LED con el pitido periódico emitido por un buzzer (similar a las alarmas de algunos vehículos), de modo que notifique de un posible estado de alerta como consecuencia de un valor extremo registrado en la temperatura.

A continuación trataremos de simular este sistema, y si tenemos éxito, lo integraremos posteriormente en el programa principal como una función más del mismo, capaz de hacer saltar la alarma cuando exceda un rango de temperatura predefinido por el usuario.

Adjuntamos el esquema del montaje correspondiente al sistema. Como vemos en la figura 4.17, es recomendable colocar una resistencia de 100Ω entre el pin digital y el extremo positivo del buzzer. La otra patilla la conectamos directamente a masa (pin GND en Arduino).

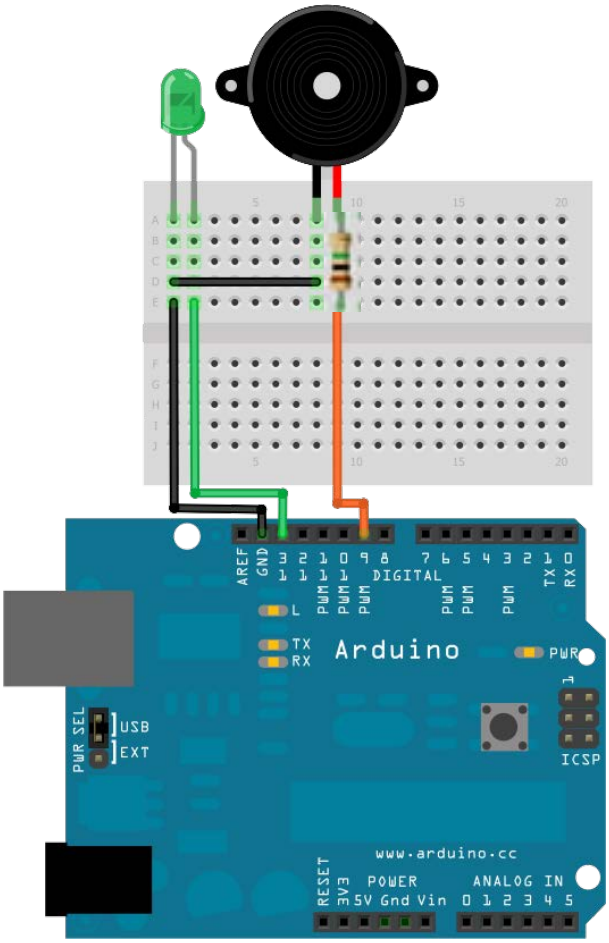


Figura 4.17: Esquema de montaje para el sistema de alarma con un Buzzer y un LED

El programa para hacer lucir un LED ya lo vimos anteriormente en el apartado "4.2. Comenzando con Arduino. Hola mundo". Ahora modificaremos dicho código para añadir el tono emitido por el buzzer, resultando el sketch adjunto en la siguiente figura 4.18:

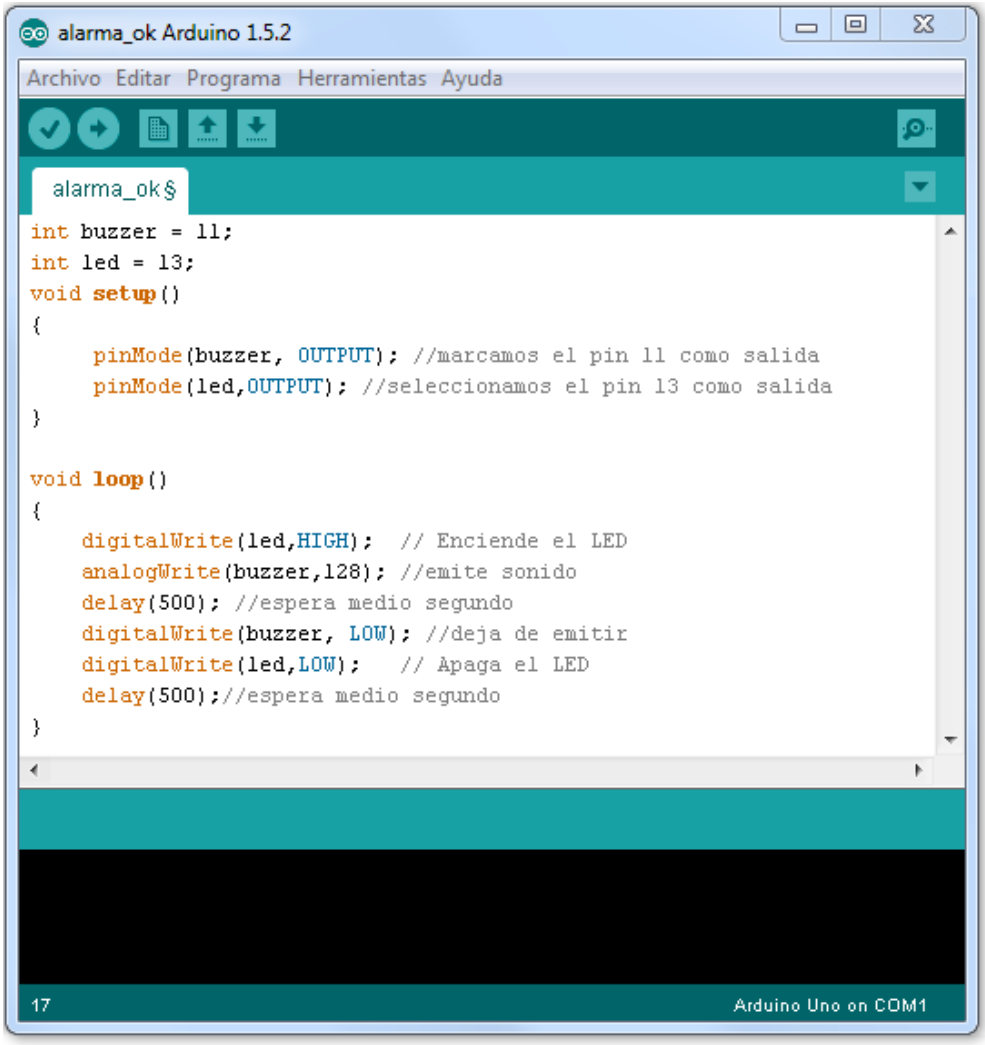


Figura 4.18: Sketch correspondiente a un posible sistema de alarma

Una vez compilado, debemos aseguramos de que tenemos conectado nuestro sistema al puerto USB del PC, y procedemos con la carga del programa en la memoria de Arduino.

Finalizado el proceso de carga, el sistema de alerta se activará de inmediato, haciendo parpadear el LED y emitiendo el pitido en sincronía a través del buzzer, todo ello en intervalos periódicos de medio segundo.

4.6. SHIELD GPRS/GSM

Por fin entramos en la parte más interesante del proyecto, donde trabajaremos con los sistemas de comunicaciones GPRS/GSM.

Recordamos que el módulo escogido para nuestro proyecto fue el "GPRS/GSM QUADBAND MODULE FOR ARDUINO (SIM900)", el cual se muestra en la figura 4.19:

Figura 4.19: Módulo GPRS/GSM (SIM900)

Junto con el módulo, se debe adquirir una antena y una fuente de alimentación externa, ya que los 5V de Arduino pueden no ser suficientes para alimentar tanto al módulo como a los componentes que conectemos a nuestra plataforma. Adquirimos también por tanto los siguientes ítems:

- Antena GPRS/GSM:

  - Frecuencia: 900 MHz-2.1 GHz-1800 MHz
  - Impedancia: 50 Ohms
  - Polarización: vertical
  - Ganancia: 0 dBi
  - VSWR: <2:1
  - Potencia: 25W
  - Conector: UFL
  - Tamaño: 35mm x 6mm
  - Temperatura de funcionamiento: de -40°C a +85°C

- Fuente de alimentación externa 18V:

  - Tensión de entrada = 100-240 V
  - Tensión de salida = 18 V_{DC}
  - Corriente máxima = 1,2 A
  - Diámetro del conector = 2,1 mm
  - Diámetro de la cubierta del conector = 5,5 mm

Una vez que tuvimos claro todos los elementos que eran necesarios para poder integrar la shield GPRS/GSM con la placa Arduino, realizamos el pedido.

A los pocos días ya contábamos con el módulo junto con el resto de accesorios, antena GPRS/GSM y fuente de alimentación.

La verdad es que desde el principio nuestra experiencia con este módulo ha pasado por varios baches importantes hasta llegar a aportar el rendimiento que se esperaba desde un primer momento.

Prácticamente desde que recibimos la placa junto con el resto de ítems que acabamos de comentar, comenzaron a surgir los problemas. Más adelante iremos entrando en detalle sobre todos los impedimentos que nos han ido surgiendo en el desarrollo práctico.

4.6.1. Configuración previa del módulo

Ilusionados por la llegada del material, nos disponemos a probar el módulo. Para ello acudimos a l a página tutorial del mismo, [18] donde podemos encontrar la operativa recomendada a seguir para evaluar su correcto funcionamiento.

NOTA: Actualmente no está disponible el tutorial que nosotros seguimos en su momento, ya que sacaron una nueva versión del módulo GPRS/GSM y actualizaron la página tutorial. Ahora los códigos que recomiendan para hacer las pruebas difieren bastantes de los que tuvimos a nuestra disposición en un principio (Febrero 2013).

El primer apartado del tutorial hace referencia al diagrama de conexiones de la shield, el cual podemos ver en detalle en la siguiente figura 4.20:

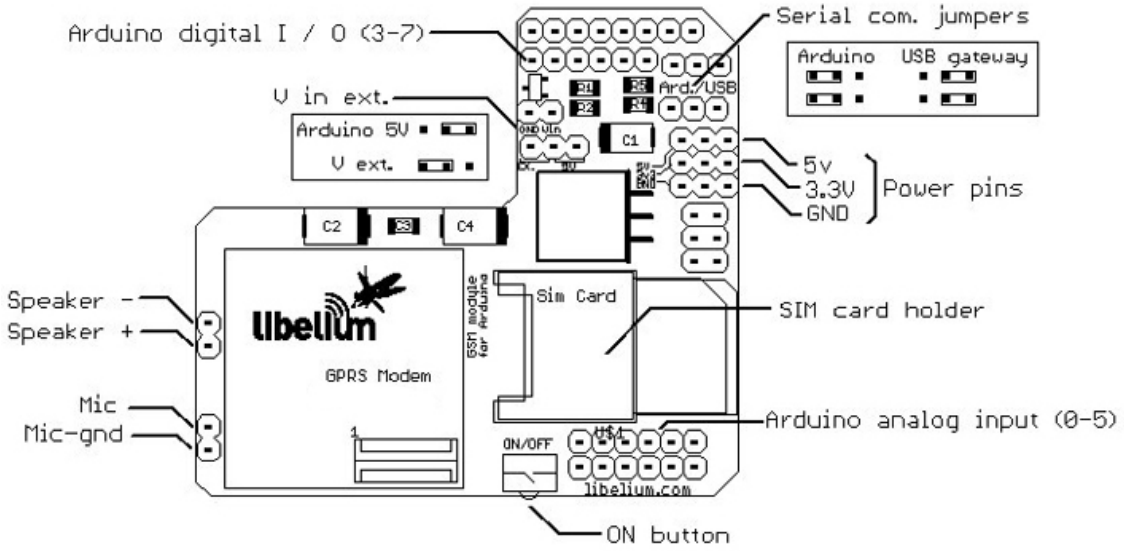
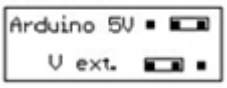


Figura 4.20: Diagrama de pines y conexiones del módulo GPRS/GSM (SIM900)

Hay que tener especial cuidado con los jumpers de la placa, uno determinará la fuente de alimentación (parte izquierda del diagrama adjunto), y los otros dos seleccionan el modo de funcionamiento de la placa (a la derecha en el mismo diagrama).

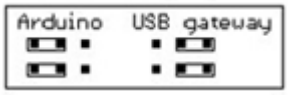
- Jumper de alimentación:



Como vemos en la imagen, se trata de un único jumper (pinza) que admite dos posiciones, "Arduino 5V" y "V ext.". Si lo colocamos en posición "Arduino 5V", el módulo recibirá la alimentación a través del pin 5V de nuestro Arduino. En cambio, si colocamos el jumper en posición "V ext.", esperará recibir la alimentación directamente por el pin "V in ext.".

No debemos relacionar el modo alimentación "V. ext" con tener o no conectada la fuente externa a nuestra placa Arduino. Con esa fuente lo único que hacemos es sobrealimentar la plataforma para que sea capaz de suministrar alimentación tanto a la shield, como a los elementos conectados a ella. El modo "V ext." es para alimentar directamente el módulo a través de una fuente externa, excluyendo de esta función al Arduino. De cualquier otro modo debemos tener el jumper en posición "Arduino 5V", para que reciba alimentación a través del mismo. Es nuestro caso, así será.

Colocamos por tanto el jumper de alimentación en modo “Arduino 5V”, y nos despreocupamos de esto. No tendremos que moverlo de ahí para nada.

- Jumpers para determinar el modo de funcionamiento: 

En esta ocasión se trata de dos jumpers con los que podemos seleccionar dos modos distintos de configuración para nuestro módulo, “Arduino” y “USB gateway”.

Comentamos primero el modo “USB gateway”. Cuando seleccionamos este modo, Arduino se convierte únicamente en una plataforma de paso de datos, es decir, hace de puente de para la comunicación entre la shield GPRS/GSM y el puerto USB. Este modo es el que debemos seleccionar para comunicarnos con el módulo por medio de comandos AT. De la misma manera, colocaremos los jumpers en esta posición cada vez que vayamos a cargar un programa en la memoria de Arduino. De no ser así el programa no se cargará. Otra opción es desconectar la shield de la plataforma, de ese modo también podremos cargar un sketch en nuestro Arduino.

Modo “Arduino”. Colocaremos los jumpers en esta posición siempre nuestra intención sea que un programa previamente cargado en la memoria de Arduino se ejecute de manera automática, es decir, sin necesidad de tener que teclear manualmente por consola los comandos AT. En nuestro caso, cuando queramos enviar un SMS a través de un programa previamente cargado, los jumpers de la shield deberán estar situados en posición “Arduino”, de no ser así, el módulo hará caso omiso de las instrucciones que le lleguen a través del Arduino.

Es muy importante tener claros todos estos conceptos referentes a los jumpers de la shield GPRS/GSM, pues será lo que más problemas pueda acarrearnos a la hora de interactuar con la placa.

Una vez visto el conexionado de la placa, pasamos al montaje de la misma sobre la plataforma Arduino UNO.

Conectamos la antena GPRS/GSM a nuestra shield, e introducimos la tarjeta SIM en su ranura correspondiente. Para mayor comodidad, recomendamos que quitéis previamente el PIN de vuestra tarjeta SIM, así evitaremos tener que introducir un comando o instrucción específica para habilitarla.

Por último, insertamos el módulo sobre la placa Arduino. Podemos ver el montaje resultante en la siguiente figura 4.21:

Figura 4.21: Plataforma Arduino UNO + Shield GPRS/GSM

A partir de este momento, sólo tendremos que ir cambiando la posición de los jumpers en función de la tarea que queramos ejecutar, seleccionando los distintos modos de funcionamiento según corresponda.

4.6.2. Envío de SMS por comandos AT

A continuación, procedemos con la primera de las pruebas recomendadas por el fabricante del módulo GPRS/GSM, el envío de un SMS mediante comandos AT.

Lo primero que debemos plantearnos a la hora de llevar a cabo cualquier prueba con el módulo es la configuración de sus jumpers. En este caso, como queremos comunicarnos con la shield directamente sin necesidad de cargar ningún código en la memoria de Arduino, colocaremos los jumpers en modo "USB gateway", tal y como se muestra en la figura 4.22.

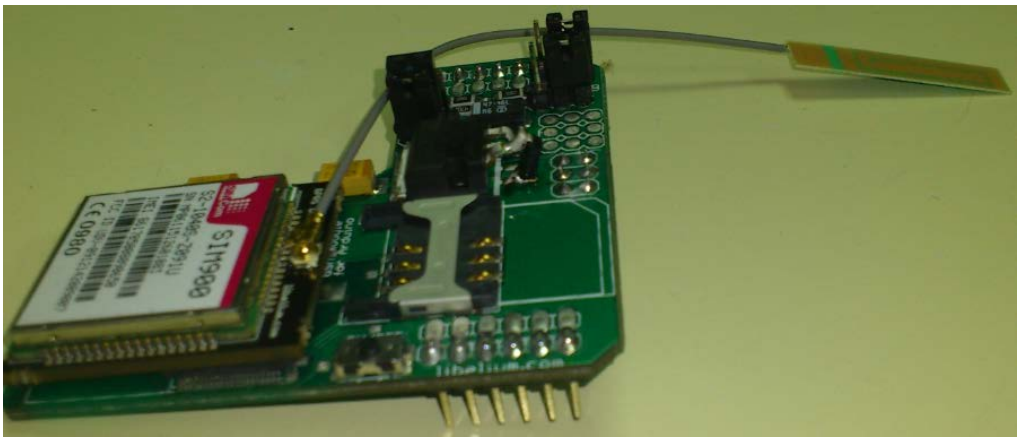


Figura 4.22: Colocación de los jumpers en modo “USB gateway”

Una vez que tenemos colocados los jumpers en modo “gateway”, montamos el módulo sobre la placa Arduino y lo conectamos al puerto USB de nuestro PC.

Llegados a este punto debemos abrir el IDE de Arduino y cargar un código vacío sobre el mismo, de manera que este no haga nada y simplemente actúe como puente para la comunicación entre el módulo GPRS y el PC. En la figura 4.23 podemos ver dicho código.

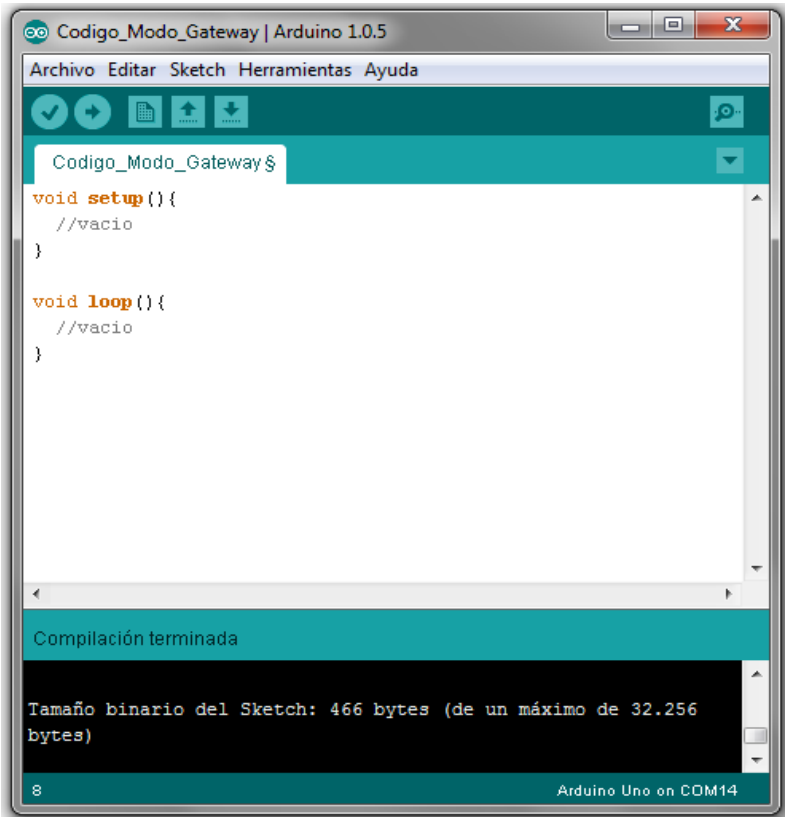


Figura 4.23: Código para la configuración del módulo GPRS/GSM en modo gateway

Una vez que hemos cargado el código, abrimos el "Monitor Serial"  . A continuación vamos a la pestaña de selección de caracteres de fin de línea, y clicamos sobre la opción "Ambos NL & CR", tal y como se muestra en la figura 4.24. Si no configuramos esto correctamente, las respuestas del módulo ante los comandos AT no aparecerán por pantalla.

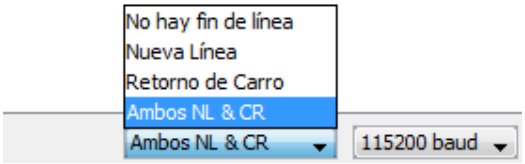


Figura 4.24: Ventana de selección de caracteres de fin de línea

El siguiente paso es encender el módulo. Para ello mantenemos presionado el botón de encendido de la shield durante 2 segundos y comenzamos el intercambio de comandos AT, obteniendo como resultado de la comunicación el que refleja la figura 4.25:

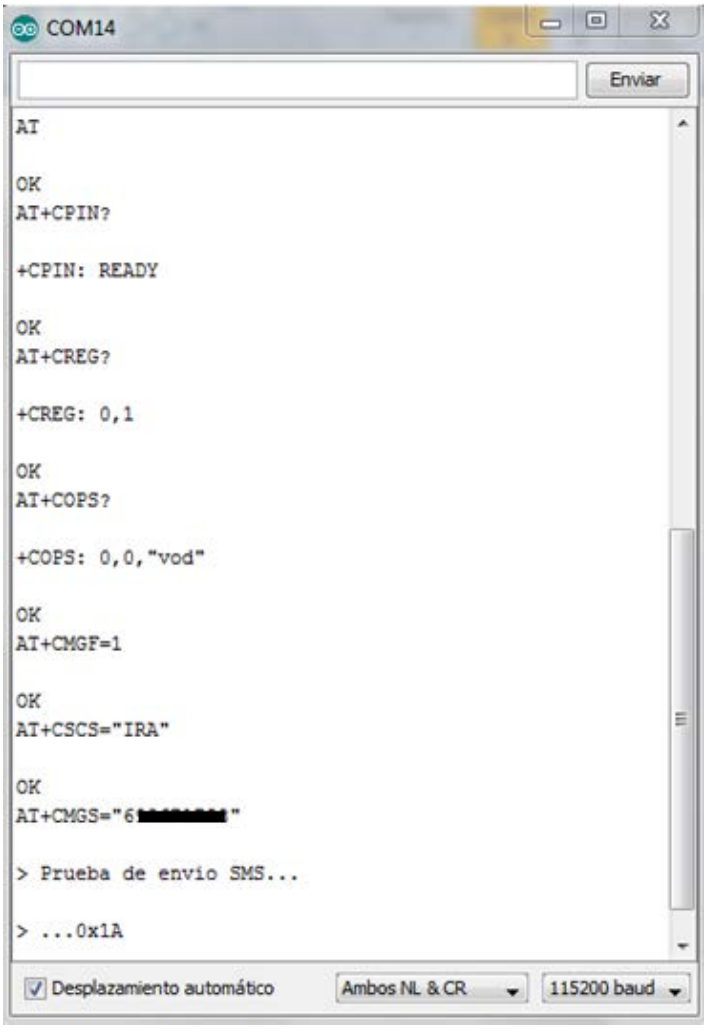


Figura 4.25: Monitor Serial durante el intercambio de comandos AT para el envío de SMS

Como vemos en la imagen, comenzamos enviando el comando “AT” para comprobar que el módulo responde correctamente ante nuestras instrucciones, si es así, responderá con un “OK”.

A continuación preguntamos si es necesario introducir el código PIN en la SIM mediante el comando “AT+CPIN?”. En este caso, como hemos quitado previamente el PIN de nuestra tarjeta, el módulo nos responde que ya está listo para operar sin necesidad de introducir el PIN manualmente.

Tras esto, comprobamos el estado de la red, para ver si ya ha sido registrado y está habilitado para el envío de SMS. Por indagar un po co más, introducimos el comando “AT+COPS?” para ver si responde correctamente con el nombre de la compañía telefónica a la que pertenece nuestra SIM (en nuestro caso Vodafone).

Lo siguiente es seleccionar el modo para el envío de SMS. Para ello tecleamos los comandos “AT+CMGF=1” y “AT+CSCS=”IRA””.

Una vez recibidos los “OK” de respuesta, podemos pasar a introducir el número al que queremos enviar el SMS y el texto que incluiremos en el mismo. Para el número escribimos el comando “AT+CMGS=”*numero móvil*””, e introducimos el texto a continuación, eso sí, a la hora de enviar el texto del SMS debemos seleccionar la opción “No hay fin de línea” en lugar de “Ambos NL & CR” como teníamos seleccionado hasta este punto. (Véase la figura 4.26)

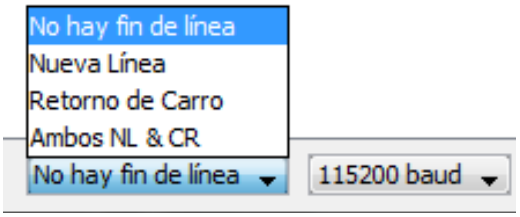


Figura 4.26: Ventana de selección de caracteres de fin de línea

Por último, enviamos el carácter Hexadecimal “0x1A”, con el cual indicamos el fin del mensaje. Inmediatamente después, el módulo procederá a su envío, y transcurridos unos segundos el SMS enviado llegará a nuestro móvil, con el texto “Prueba de envío SMS…”.

Como todo ha salido según lo esperado y hemos recibido correctamente el SMS en nuestro móvil, pasamos ahora a hacer la prueba del envío de un S MS pero esta vez, a través de Arduino, sin necesidad de teclear manualmente los comandos AT.

4.6.3. Envío de SMS a través de Arduino

Para poder comunicarnos con el módulo GPRS/GSM a través de Arduino y conseguir enviar un SMS tendremos que generar un código compuesto por las mismas instrucciones (comandos AT) que le enviábamos en el apartado anterior, de manera que cuando carguemos dicho programa en la memoria de nuestro Arduino éste se encargue de transmitirle esos comandos en el mismo orden que lo haríamos nosotros manualmente.

Como siempre, lo primero que debemos hacer es plantearnos la colocación que deben tener los jumpers del módulo. En este caso el proceso es un poco más complicado que el anterior, pues para que la shield ejecute las órdenes procedentes de Arduino los jumpers deben estar colocados en modo "Arduino", pero sin embargo, para poder cargar un programa (sketch) cuando la shield GPRS está montada sobre él, es necesario que los jumpers se encuentren en modo "USB gateway". Por lo tanto, a la hora de cargar el programa lo haremos con los jumpers en modo "gateway", y una vez cargado tendremos que cambiarlos a la posición "Arduino" para que el módulo atienda las instrucciones procedentes del mismo.

Los pasos a seguir serán:

1) Generar el sketch con los comandos AT adecuados para que el módulo envíe un SMS. Adjuntamos el código del programa:

```
int led = 13;                //pin correspondiente al led
int onModulePin = 2;         // pin correspondiente al botón de encendido
                             // para no tener que encenderlo manualmente
int timesToSend = 1;         // cuantos mensajes queremos enviar
int count = 0;

void switchModule()
{
    //equivale a presionar 2 segundos el botón de encendido
    digitalWrite(onModulePin,HIGH);
    delay(2000);
    digitalWrite(onModulePin,LOW);
}

void setup(){
    Serial.begin(19200);   // baudrate
    delay(2000);
    pinMode(led, OUTPUT);
    pinMode(onModulePin, OUTPUT);
    switchModule();        // para encender el módulo automáticamente
}
```

```
    for (int i=0;i < 5;i++){
          //tiempo de espera para asegurarnos de tener conexión a la red
          //hacemos parpadear el led para saber que está en espera
          digitalWrite(led,HIGH);
          delay(1500);
          digitalWrite(led,LOW);
          delay(1500);
    }
    Serial.println();
    Serial.println("AT+CMGF=1");  // seleccionamos el modo de texto
    Serial.println(Serial.read());
    delay(1000);
}

void loop(){
    while (count < timesToSend){
        delay(1500);
        Serial.println("AT+CMGS=\"6********\""); // móvil al que se enviará el SMS
        delay(1000);
        Serial.print("Texto SMS");     // Texto del mensaje
        delay(500);
        Serial.write(0x1A);       //indicamos fin de SMS
        delay(5000);
        count++;
        digitalWrite(led,HIGH); //señal de que el programa ha finalizado
    }
}
```

2) Nos aseguramos de que los jumpers se encuentran en modo “USB gateway”.

3) Conectamos la plataforma compuesta por Arduino y el módulo GPRS al puerto USB del ordenador.

4) Cargamos el sketch en la memoria Arduino y lo desconectamos del PC.

5) Modificamos la posición de los jumpers a modo “Arduino”.

6) Conectamos la plataforma a la fuente de alimentación externa, obteniendo como resultado el montaje que vemos en la figura 4.27.

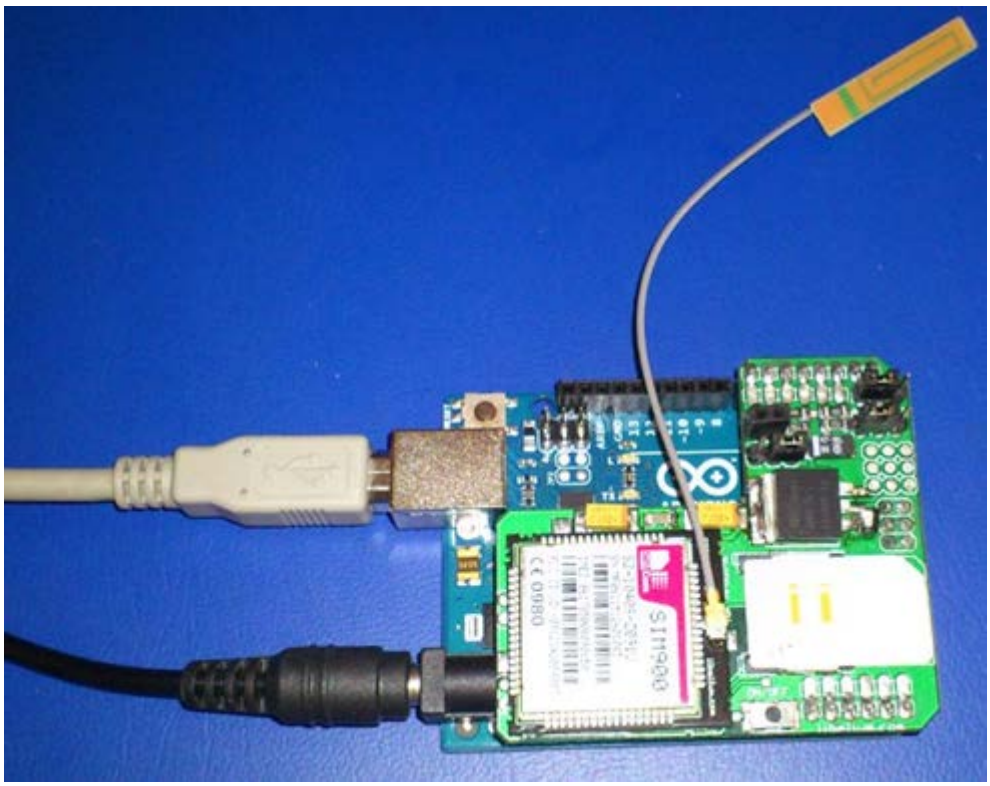


Figura 4.27: Esquema de montaje para el envío de SMS a través de Arduino

NOTA: Si queremos podemos conectarlo también al puerto USB del PC y abrir el Monitor serial para ver la ejecución del programa y observar cómo se van enviando los comandos AT.

7) El programa comenzará a ejecutarse, y si todo ha ido bien, tras unos segundos recibiremos el SMS en nuestro móvil, tal y como se muestra en la figura 4.28.

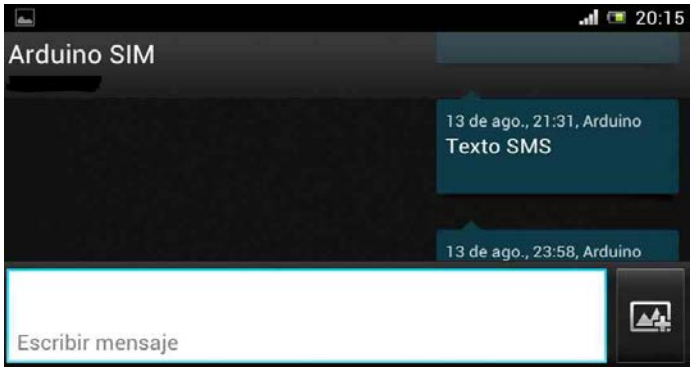


Figura 4.28: Recepción del SMS en el teléfono móvil

4.6.4. Lectura de SMS por comandos AT

En los dos próximos apartados se avaluará el comportamiento del módulo GPRS ante los comandos para la lectura de SMS almacenados en la memoria de nuestra tarjeta SIM.

En esta ocasión, comenzaremos por enviar directamente los comandos AT a través del interfaz de Arduino. Para ello, lo primero que haremos será colocar los jumpers de nuestra shield en modo “USB gateway”:

A continuación, procedemos a conectar la plataforma al puerto USB del ordenador, y alimentamos también a través de la fuente de alimentación, para asegurarnos de que no se produzcan apagones del módulo.

Cargamos el sketch con el correspondiente código vacío, simplemente con la declaración de las funciones ‘setup’ y ‘loop’, las cuales debemos incluir siempre, independientemente del programa que estemos desarrollando. Si no es así, al compilar nos dará un error, y no podremos abrir el Monitor Serial para intercambiarnos comandos AT entre el puerto USB del PC y la plataforma Arduino. En la figura 4.29 podemos ver el contenido de dicho sketch.

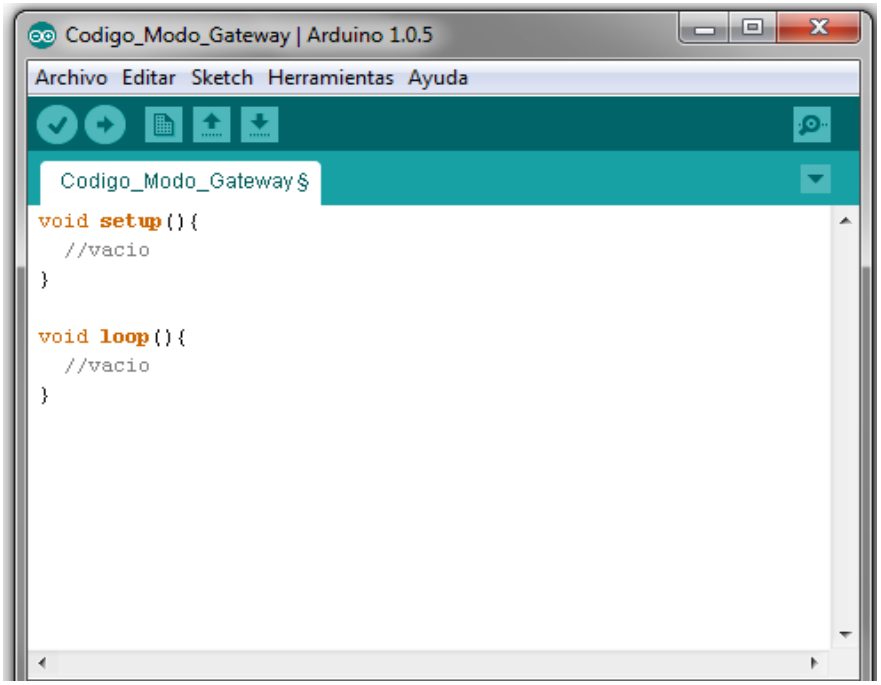


Figura 4.29: Sketch para la configuración del módulo en modo gateway

A partir de este momento, ya tenemos lista la plataforma para comenzar con el intercambio de instrucciones a través del interfaz de Arduino. Lo único que debemos hacer ahora es abrir la ventana correspondiente al Monitor Serial, seleccionar la opción para el envío de caracteres de fin de línea y retorno de carro ("*Ambos NL & CR*") y comenzar a transmitir los comandos.

Los pasos que seguiremos para la lectura de un SMS desde la memoria de la tarjeta SIM se detallan a continuación. En la figura 4.30 se muestran los resultados.

1. Encendemos manualmente la shield GPRS. Para ello mantenemos pulsado el botón de encendido durante 2-3 segundos. Por pantalla veremos que nos aparece una serie de 8 caracteres por defecto correspondientes al encendido del módulo, siempre es así, debemos esperar a que termine dicha cadena y entonces estará listo para atender nuestras instrucciones.

2. Como de costumbre, enviaremos un simple comando "AT" para ver si el módulo responde correctamente y atiende a los datos que se le envían por el puerto serie. Si es así, responderá con un "OK".

3. A continuación seleccionaremos el formato del mensaje a través del comando "AT+CMGF=1", indicando que se trata de manejar SMS en modo texto. Si ha interpretado el comando correctamente nos responderá de nuevo con un "OK".

4. El siguiente paso es quizá el más importante, ya que debemos indicar la memoria de la que pretendemos leer el SMS en cuestión, en nuestro caso la de la tarjeta SIM (siglas "SM"). Para ello debemos incluir la siguiente instrucción "AT+CPMS="SM"". La respuesta por parte del módulo debe ser "+CPMS: <x>, <y>", donde <x> indica el número de SMS almacenados en la memoria, e <y> hace referencia al número máximo de mensajes que se pueden llegar a ella. Como vemos en la figura 4.30, en un principio nosotros no tenemos ningún mensaje, pero nuestra SIM es capaz de almacenar hasta 20 mensajes de texto (*0, 20*). En ese momento enviamos un SMS a la SIM insertada en el módulo, y de inmediato recibimos la notificación correspondiente a su recepción a través de la instrucción +CMTI. En consecuencia, si volvemos a preguntar por el estado de la memoria de nuestra tarjeta SIM (véase la figura 4.30), ahora ya nos indica que tenemos 1 SMS *(1, 20*), que será el que leamos en esta prueba.

5. Una vez que nos hemos asegurado de que ya tenemos mensajes disponibles en la memoria, procedemos a su lectura a través del comando "AT+CMGR=<pos>", donde en el lugar de <pos> debemos incluir la posición que ocupa el mensaje en la

memoria de la tarjeta. En nuestro caso no cabe duda de que debemos seleccionar la posición 1, ya que es el único SMS que contiene la memoria, pero a medida que recibiéramos más SMS, estos se irían almacenando en posiciones continuas, es decir, un s egundo mensaje iría a la posición 2, e l siguiente a la 3, y así sucesivamente.

Tras enviar este último comando para la lectura del mensaje, la shield nos escribirá por pantalla el contenido del SMS ("*Leyendo mensaje por comandos AT*"), precedido por la respuesta "+CMGR" y una serie de información relacionada con el mensaje, como la hora y fecha de envío, si el mensaje ya había sido leído previamente o no, y el número del remitente. Para terminar mostrará por pantalla un "OK", indicando que ha finalizado con el proceso de lectura del mensaje, y podremos dar por concluida esta prueba.

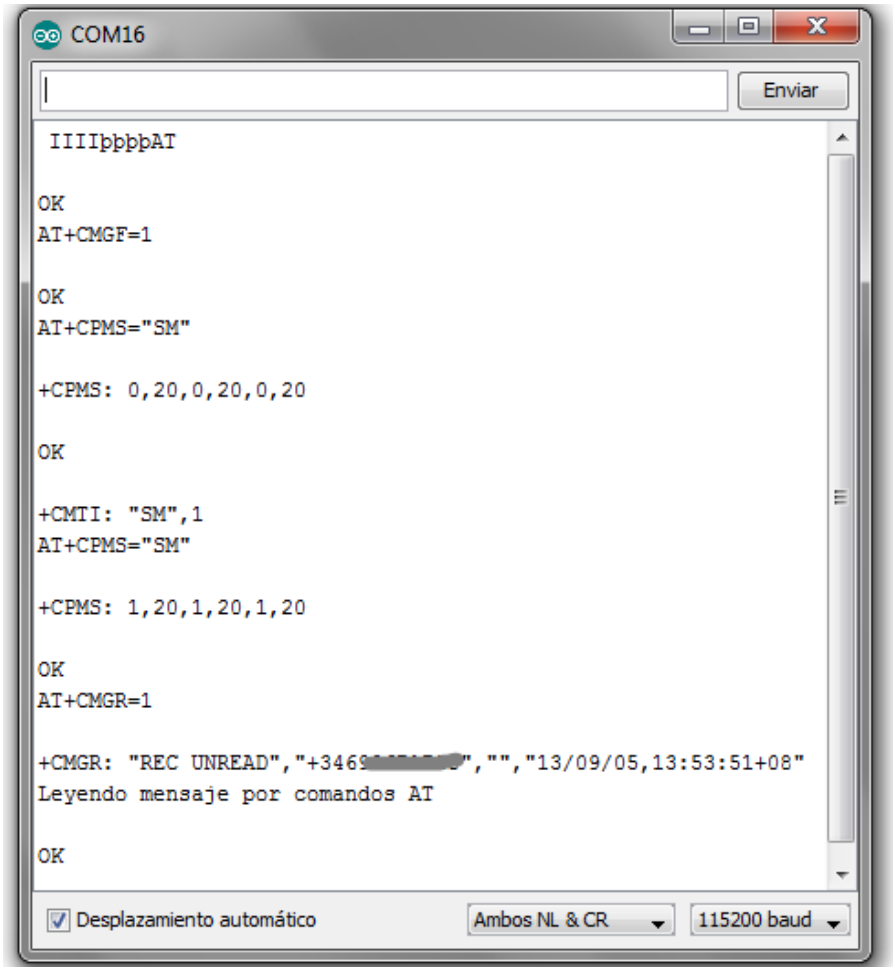


Figura 4.30: Intercambio de comandos AT para la lectura de un SMS

De momento todo ha estado funcionando correctamente, y hemos conseguido con éxito llevar a cabo las pruebas de envío y lectura de SMS a través de comandos AT, pero pronto empezarán a surgir los problemas que adelantábamos en relación al módulo GPRS/GSM.

4.6.5. Lectura de SMS a través de Arduino

Entramos en la última de las pruebas que teníamos intención de realizar antes de adentrarnos en el desarrollo del programa general para nuestra aplicación. Se trata de leer un SMS que se encuentre en la tarjeta SIM a través de un sketch cargado previamente en la memoria de nuestro Arduino.

Llegados a este punto, se pretende utilizar el código que aparece a modo de ejemplo en la página tutorial de la shield GPRS/GSM. [18] Dicho código viene preparado para comunicarse con el módulo a la vez que tenemos abierta la ventana correspondiente al Monitor Serial, de manera que podemos ver por pantalla los comandos que Arduino envía al módulo y las respuestas del mismo, así como el contenido del mensaje leído.

En el momento en que se llevaron a cabo las pruebas con este código, nos dimos cuenta de que nuestra shield GPRS no respondía ante los comandos AT cuando teníamos colocamos los jumpers en modo 'Arduino'. Ante los mismos comandos con los que antes hemos conseguido leer un SMS sin ningún problema, ahora no responde, y por lo tanto el programa no funciona.

En seguida comprendimos que nuestro módulo tenía algún tipo de problema, y que sería necesario ponernos en contacto con el servicio técnico de Libelium (empresa a la que compramos el material). Como era de esperar, tuvimos que enviar el módulo para que fuera examinado y pudiesen darnos una solución a nuestro problema.

Tras unos días en los que evaluaron el comportamiento de la shield, nos confirmaron que efectivamente la placa presentaba algún defecto de fabricación, por el cual el módulo no respondía ante los comandos cuando estamos trabajando en modo 'Arduino'. Como solución, nos enviaron un módulo completamente nuevo, y además, como se había fabricado una nueva versión del mismo, nos hicieron llegar la shield GPRS correspondiente al último modelo, el cual mostramos en la figura 4.31.

Figura 4.31: Nueva shield GPRS/GSM (SIM900)

Como se ve en la imagen, este nuevo modelo apenas difiere su versión anterior. Los cambios más significativos se resumen en un leve cambio en su diseño, la integración de un LED que nos indica cuándo esta operativa la shield, y un anclaje más robusto para el módulo SIM900, que al contrario que en la placa antigua, ahora viene soldado completamente a la base, mientras que antes solo se anclaba en uno de los extremos, y lo hacía algo inestable. Además, tal como se muestra en la figura 4.32, este modelo incluye un jumper específico para la conexión con Raspberry Pi, algo interesante para la segunda fase del proyecto, en la que mi compañero tiene la intención de utilizar este pequeño ordenador como plataforma para la conexión a internet del sistema, permitiendo la posibilidad de subir datos referentes a sensores a un s ervidor web. En mi caso, este jumper se mantendrá siempre en posición Arduino durante el desarrollo del proyecto.

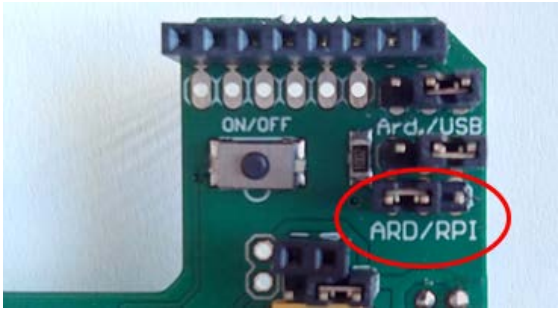


Figura 4.32: Jumper para la selección de modo Arduino/Raspberry Pi

Desde el momento que recibimos este nuevo módulo GPRS las cosas comenzaron a funcionar mucho mejor. Probamos de nuevo los programas para envío y lectura de SMS, y en esta ocasión no obtuvimos ningún problema.

A continuación vamos a an alizar el código correspondiente al sketch para la lectura de mensajes, el cual se adjunta en la figura 4.33. Para verlo en detalle, vamos a ir comentando brevemente cada una de las funciones de las que se compone el código.

Comenzamos por la declaración de variables y la función ‘setup’. Esta función es la primera en ejecutarse tras el arranque del programa. Lo primero que hace es establecer el valor de baudrate (tasa de transmisión). A continuación, se enciende el módulo de manera automática, evitando que tengamos que hacerlo manualmente como ocurre cuando intercambiamos comandos AT. Para ello vemos que llama a la función ‘power_on’, cuya misión consiste, primero en comprobar si el módulo ya ha sido encendido previamente, y segundo actuar en consecuencia, encendiendo la shield si es necesario.

Una vez que se ha encendido la shield, comienza el envío de los comandos AT a través de la función 'sendATcommand'. A esta función debemos pasarle tres parámetros, el comando que queremos enviar, la respuesta esperada en el caso de que el comando se reciba correctamente, y un tiempo determinado para reenviar en varias ocasiones el comando en caso de que la respuesta no sea la esperada.

Los comandos AT que se envían son exactamente los mismos que veíamos en el apartado anterior ("4.6.4. Lectura de SMS por comandos AT"), solo que esta vez es el propio Arduino quien envía las instrucciones al módulo GPRS, en lugar de tener que teclearlos el usuario manualmente. No merece la pena volver a entrar en detalle sobre estos comandos, pues ya se ha detallado su función en el apartado anterior.

Tras enviar el comando correspondiente a la lectura del primer mensaje almacenado en la memoria de la SIM ("AT+CMGR=1"), se incluye un algoritmo para almacenar el contenido del SMS en una cadena de caracteres, la cual será impresa a continuación por pantalla.

```
int8_t answer;
int x;
int onModulePin= 2;
char SMS[200];

void setup(){

    pinMode(onModulePin, OUTPUT);
    Serial.begin(115200);

    Serial.println("Encendiendo el modulo GPRS...");
    power_on();

    delay(3000);

    Serial.println("Configurando para leer un SMS...");
    sendATcommand("AT+CMGF=1", "OK", 1000);    // modo texto
    sendATcommand("AT+CPMS=\"SM\",\"SM\",\"SM\"", "OK", 1000);  // memoria de tarjeta SIM

    answer = sendATcommand("AT+CMGR=1", "+CMGR:", 2000);  // leemos el primer SMS en la memoria
    if (answer == 1)
    {
        answer = 0;
        while(Serial.available() == 0);
        // algoritmo para lectura del SMS
        do{
            // mientras halla datos que leer los almaceno en la cadena llamada SMS
            if(Serial.available() > 0){
                SMS[x] = Serial.read();
                x++;
                // cuando se lea el OK de fin de mensaje paramos de leer
                if (strstr(SMS, "OK") != NULL)
                {
                    answer = 1;
                }
            }
        }while(answer == 0);
```

```
        SMS[x] = '\0';
        Serial.print(SMS);
    }
    else
    {
        Serial.print("error ");
        Serial.println(answer, DEC);
    }
void loop(){
  // vacio
}

void power_on(){

    uint8_t answer=0;

    // comprobamos que no este ya encendido previamente
    answer = sendATcommand("AT", "OK", 2000);
    if (answer == 0)
    {
        // encendemos
        digitalWrite(onModulePin,HIGH);
        delay(3000);
        digitalWrite(onModulePin,LOW);

        // esperamos hasta comprobar que se ha encendido y responde
        while(answer == 0){   // Enviamos el comando AT cada dos segundos
            answer = sendATcommand("AT", "OK", 2000);
        }
    }
}
int8_t sendATcommand(char* ATcommand, char* expected_answer, unsigned int timeout){
    uint8_t x=0,  answer=0;
    char response[100];
    unsigned long previous;

    memset(response, '\0', 100);    // Dejamos limpia la cadena
    delay(100);
    while( Serial.available() > 0) Serial.read();    // Limpiamos el buffer
    Serial.println(ATcommand);    // Enviamos el comando
    x = 0;
    previous = millis();

    do{  // loop esperando una respuesta
        //mientras tenemos datos en el buffer los leemos
        if(Serial.available() != 0){
            response[x] = Serial.read();
            x++;
            // comprobamos que la respuesta es la esperada
            if (strstr(response, expected_answer) != NULL)
            {
                answer = 1;
            }
        }// Esperamos un cierto tiempo a que responda

    }while((answer == 0) && ((millis() - previous) < timeout));
    return answer;
}
```

Figura 4.33: Sketch correspondiente al código para lectura de un SMS

A continuación vamos a detallar los pasos a seguir para conseguir leer un SMS a partir de un programa cargado previamente en la memoria de nuestro Arduino.

1. En primer lugar procedemos con la carga del programa. Para ello, como ya se ha comentado otras veces en apartados anteriores, debemos conectar la plataforma compuesta por Arduino y la shield GPRS a nuestro PC, y configurar los jumpers en modo 'USB gateway' para la carga del sketch, tal como se muestra en la siguiente figura 4.34:

Figura 4.34: Colocación de los jumpers operando en modo "USB gateway"

2. Una vez que el código ha sido cargado correctamente, debemos cambiar la configuración de los jumpers y colocar los mismos en posición 'Arduino'. Acto seguido alimentamos también la plataforma con la fuente de alimentación externa, y abrimos el Monitor Serial para observar el programa en ejecución. En la figura 4.35 se muestra el montaje del sistema, donde se detalla la posición de los jumpers.

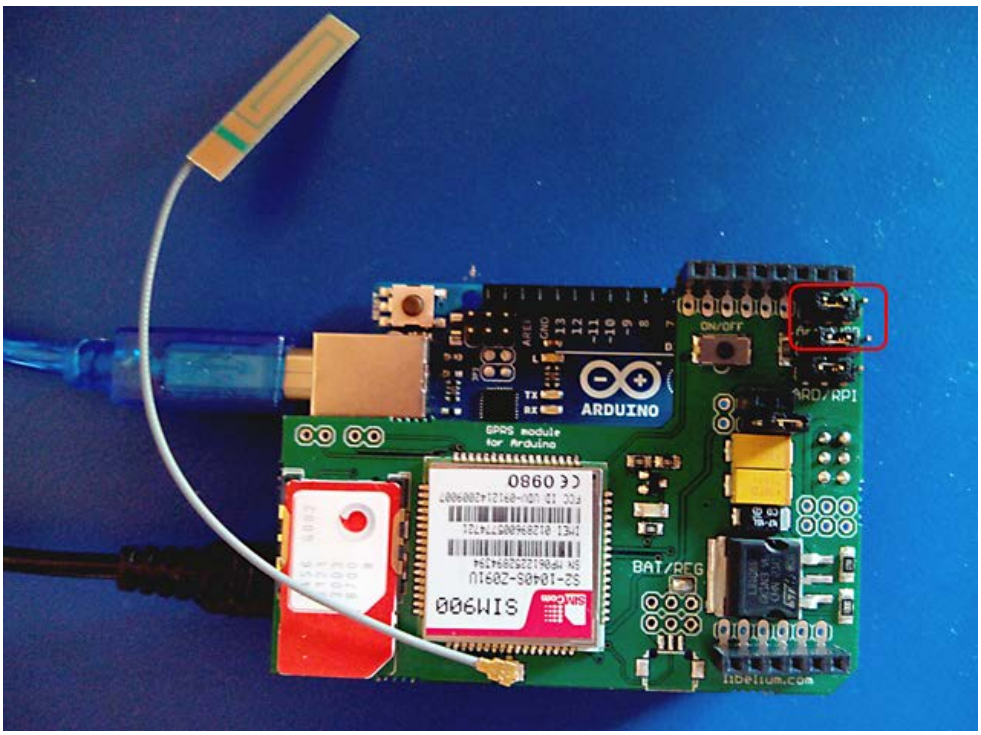


Figura 4.35: Conexión de la plataforma con los jumpers en modo Arduino

3. Ahora solo tenemos que esperar a que aparezcan por pantalla los resultados correspondientes al desarrollo del programa, y por consiguiente, el contenido del SMS que se ha leído, en este caso "*Leyendo mensaje a través de Arduino*". En la siguiente figura podemos ver la pantalla del Monitor Serial tras la ejecución del programa.

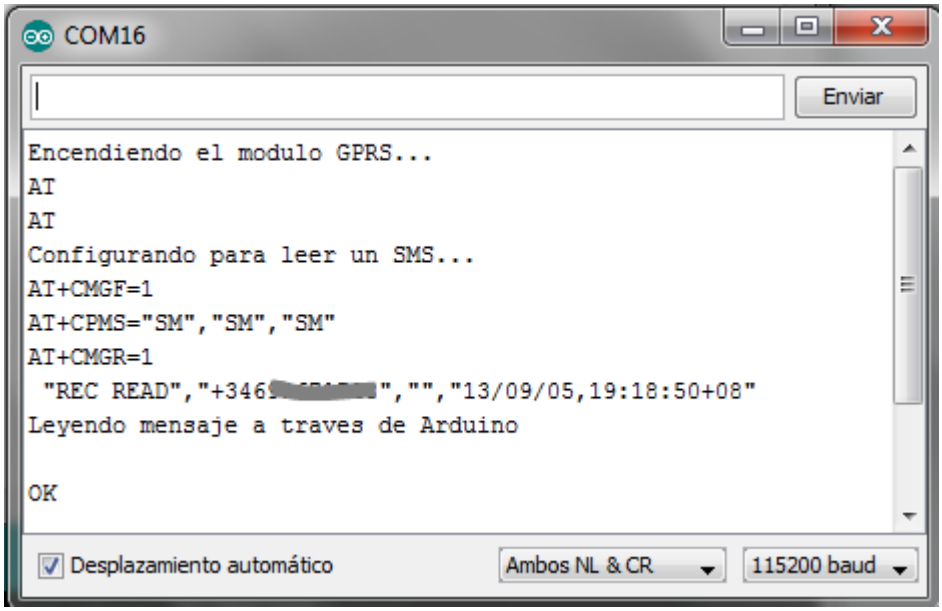


Figura 4.36: Monitor Serial tras la lectura de un SMS a través de Arduino

Llegados a este punto, damos por finalizadas las pruebas referentes al comportamiento del módulo GPRS/GSM, y pasaremos a la integración de las distintas funciones en el programa correspondiente a la aplicación general para el control de la temperatura.

4.7. DESARROLLO DE APLICACIONES Y EVALUACIÓN DEL SISTEMA

Por fin nos adentramos en la fase final del proyecto, donde se llevará a cabo el desarrollo de las aplicaciones para la implementación del sistema.

En primer lugar se va a diseñar una aplicación capaz de integrar las distintas funcionalidades evaluadas hasta el momento (sensor de temperatura, sistema de alarma, envío de mensajes, y accionamiento de un motor servo). Esta primera aplicación tendrá como plataforma de gestión al propio IDE de Arduino, donde aparecerá un menú que nos permita seleccionar en cada momento las distintas aplicaciones que queremos ejecutar.

En segundo lugar se implementará otra aplicación que nos permita consultar el valor de temperatura y actuar en consecuencia sobre el servomotor de un posible termostato, pero esta vez la gestión se hará de forma remota, a través del envío y recepción de SMS. Este sistema podrá estar completamente desatendido (modo 'standalone'), y sólo llevará a cabo alguna acción cuando reciba la orden correspondiente a través de un mensaje de texto.

Tras el desarrollo de cada una de las aplicaciones procederemos con su evaluación, y expondremos los resultados con detalle en los apartados correspondientes (4.7.2 y 4.7.4).

4.7.1. Código de la aplicación general para el sistema de control de temperatura de gestión local

Comenzamos con el desarrollo del programa correspondiente a la aplicación general, cuya plataforma de gestión será un menú que aparecerá por la propia pantalla del entorno de trabajo de Arduino, por lo que el sistema necesitará permanecer conectado al PC en todo momento.

El material del que disponemos para la implementación del sistema se compone de:

- Arduino MEGA (* a continuación se justificará el porqué del uso de este modelo)
- Shield GPRS/GSM (SIM900)
- Sensor de temperatura con cabeza de acero (SEN118A2B)
- Micro Servo 9G
- LED
- Buzzer

* A continuación se explica el porqué de utilizar Arduino MEGA para el desarrollo del sistema. A pesar de que para las pruebas realizadas hasta el momento venimos utilizando el modelo Arduino UNO, para el diseño del sistema final se ha tomado la decisión de utilizar Arduino MEGA por una serie de justificaciones. En primer lugar, la memoria de programa del modelo UNO (32 KB) es bastante limitada como para permitir el desarrollo de una aplicación en la que ya integremos varias funcionalidades. Arduino MEGA en cambio, dispone de hasta 256 KB para memoria de programa, con lo que tendremos un margen mucho mayor a la hora de generar un sketch más completo. Otra de las razones es que con Arduino MEGA dotamos al sistema de 50 pi nes adicionales en comparación de los que ofrecía Arduino UNO, facilitando la conexión de los diferentes elementos y ofreciendo la posibilidad de escalar las prestaciones del proyecto en un futuro, conectando varios sensores o actuadores. Además, el consumo de ambas placas es el mismo, y dado que Arduino es una plataforma de hardware

libre, podemos contrarrestar el único contratiempo que supone decantarnos por Arduino MEGA, su precio. Podemos encontrar placas idénticas a Arduino MEGA (sin ser de fabricación oficial) por el mismo precio que el modelo Arduino UNO (unos 25 €), con lo cual no cabe duda de que decantarnos por el modelo MEGA es la mejor opción. En la figura 4.37 podemos comparar las características entre Arduino UNO y Arduino MEGA.

UNO		MEGA	
Microcontroller	ATmega328	Microcontroller	ATmega2560
Operating Voltage	5V	Operating Voltage	5V
Input Voltage (recommended)	7-12V	Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V	Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)	Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	6	Analog Input Pins	16
DC Current per I/O Pin	40 mA	DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA	DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader	Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	2 KB (ATmega328)	SRAM	8 KB
EEPROM	1 KB (ATmega328)	EEPROM	4 KB
Clock Speed	16 MHz	Clock Speed	16 MHz

Figura 4.37: Comparativa entre los modelos Arduino UNO y Arduino MEGA

Una vez que tenemos claro todo el material que vamos a utilizar, comenzamos con el planteamiento del sketch correspondiente a la aplicación. Para ello se diseña un diagrama de flujo con los pasos que principalmente se pretende que siga el programa. A partir del mismo podremos comenzar a programar el código, siempre con la ayuda de los distintos sketches que se han ido generando en las pruebas individuales para cada uno de los ítems (obtener valor de temperatura a partir del sensor, sistema de alarma con un buzzer y led, envío de SMS, accionar un motor servo, etc.).

En la figura 4.38 se muestra el diagrama de flujo correspondiente a la aplicación:

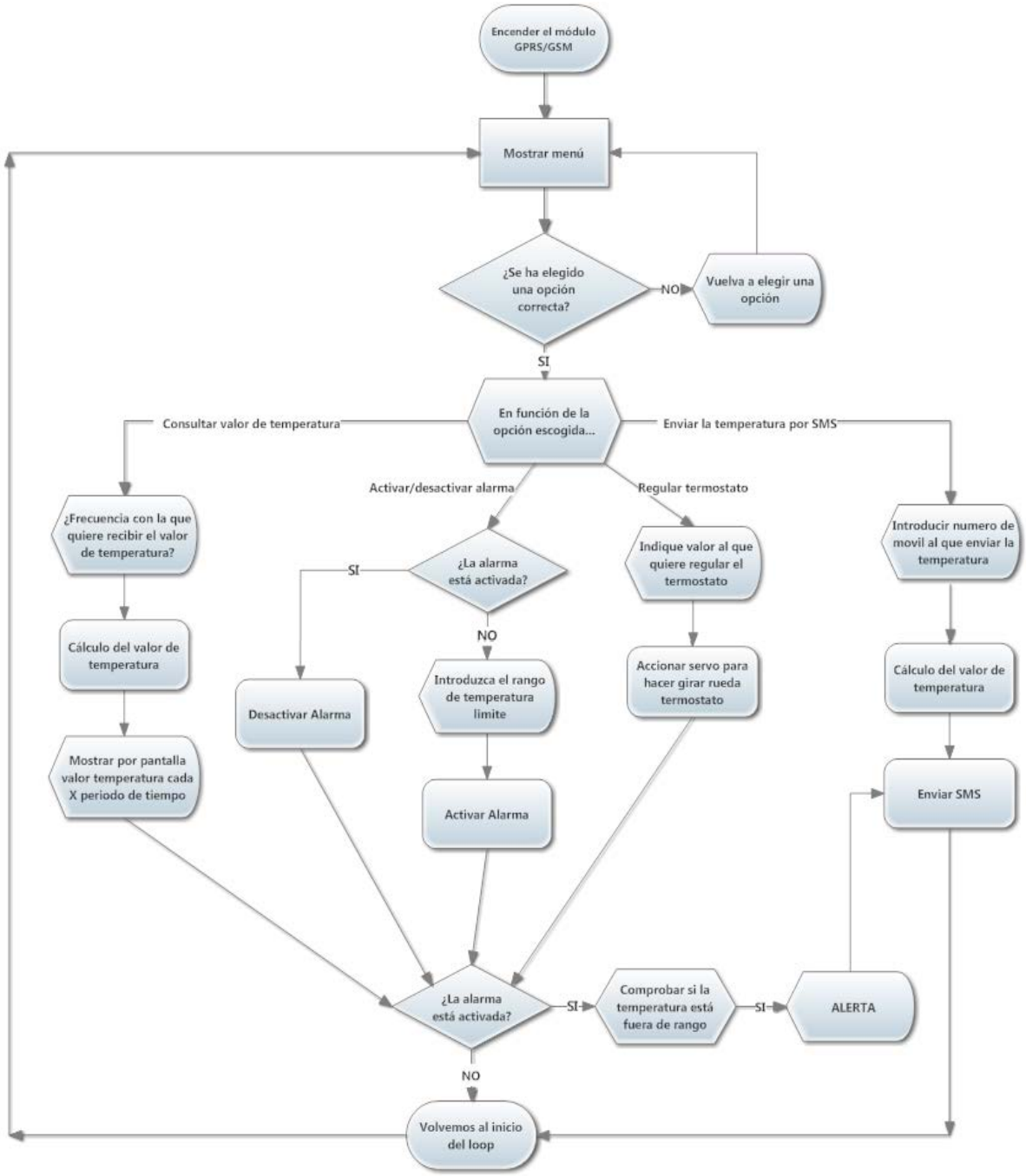


Figura 4.38: Diagrama de flujo de la aplicación general

El siguiente paso es convertir dicho diagrama de flujo en código de programación. Para ello se han tenido que ir integrando poco a poco los distintos elementos (componentes del sistema), hasta llegar a completar la plataforma en todo su conjunto. A continuación vamos a ir fragmentando el código general de la aplicación mientras vamos comentando las distintas funciones que lo componen. El resultado del sketch al completo se obtiene de la unión de cada una de las partes que se van a ir argumentando.

Comenzamos por la declaración de variables, el algoritmo para el cálculo de la temperatura y la función 'setup' con la que se inicia el programa. En la función 'setup' establecemos el valor de 'baudrate' correspondiente al puerto serie. En este caso, necesitamos dos puertos para el intercambio de datos, uno dedicado a la comunicación entre PC y la plataforma Arduino ('Serial'), y otro para la comunicación entre la placa Arduino y el módulo GPRS ('Serial1'). Dentro del 'setup' debemos configurar también como salidas los pines asociados a los diferentes actuadores (LED, botón de encendido del módulo GPRS, buzzer y servomotor). También es prioritario encender la shield GPRS/GSM al comienzo del programa, de ahí que se invoque a la función 'power_on', la cual analizaremos más adelante.

```
#include <math.h>
#include <Servo.h>

#define ThermistorPIN 0  // Analog Pin 0

int activa=0;
int aviso=0;
int maxima=0;
int minima=0;
float celsius;

Servo miServo;
int angulo=90; //eje central del servo

int8_t answer;
char phone_number[10]; //variable para guardar el numero de movil
char aux_string[30];

int buzzer = 11;
int led = 13; // pin del led
int onModulePin = 2;  // pin correspondiente a boton encendido

float vcc = 4.91;
float pad = 9850;  // valor real de la resistencia
float thermr = 10000; // valor nominal de la resistencia 10K

float Thermistor(int RawADC) {
  long Resistance;
  float Temp;
  Resistance=((1024 * pad / RawADC) - pad);
  // algoritmo para calculo de la temperatura en kelvin
  Temp = log(Resistance);
  Temp = 1 / (0.001129148 + (0.000234125 * Temp) +
         (0.0000000876741 * Temp * Temp * Temp));
  Temp = Temp - 273.15;  // Convertimos Kelvin a Celsius
  return Temp;    // Devolver el valor de temperatura en Celsius
}

void setup() {
  pinMode(onModulePin, OUTPUT);
  Serial.begin(115200); //baudrate del puerto serie con PC
  Serial1.begin(115200); //baudrate del puerto comunicacion con modulo GPRS
  pinMode(led,OUTPUT);
  pinMode(buzzer, OUTPUT);
  miServo.attach(9); //pin correpondiente al servo
  power_on(); //encendemos el modulo GPRS
}
```

Una vez que tenemos todo correctamente configurado y listo para el inicio del programa, nos centramos en la función principal, la famosa void 'loop', obligatoria en cada sketch de Arduino. Al comienzo de la misma mostraremos el menú por pantalla con las diferentes opciones que el usuario tiene a su disposición. En función de la opción escogida entraremos en el desarrollo de esa pequeña aplicación en particular, y tras llegar al final de la misma volveremos a entrar en el 'loop', a la espera de una nueva selección por parte del usuario.

```
void loop() {
  int opcion;
  int flag_alarma=0;
  int posicion;
  float grados;

  opcion = menu();
  switch (opcion) {

  case 1:
    Serial.println("\n\n  [1] Consultar temperatura:");
    consultar_temperatura();
  break;

  case 2:
    Serial.println("\n\n  [2] Enviar SMS con el valor de temperatura:");
    posicion=0;
    Serial.print("\n  Introduce el movil al que quieres que se envie el valor de temperatura:  ");
    while(Serial.available()==0){
      delay(5); //esperamos a que el usuario escriba el numero
    }
    while (Serial.available()>0){
      delay(5); //leemos el numero
      phone_number[posicion]=Serial.read();
      Serial.print(phone_number[posicion]);
      posicion++;
    }
    Serial.print("\n\n");
    delay(700);
    Serial.print("  Arduino se esta comunicando con el modulo GSM/GPRS. Por favor, espere...\n\n");
    delay(2000);
    SMS(); //procedemos al envio del SMS
  break;

  case 3:
    Serial.print("\n\n  [3] Activar/desactivar alarma por rango de temperatura:\n");
    if(activa==1){
      Serial.println("\n  La alarma ya esta activada. Pulse 0 si quiere desactivarla.");
      flag_alarma=leer();
      if(flag_alarma==0){
        Serial.println("\n  La alarma ha sido desactivada\n");
        aviso=0;
        activa=0;
        digitalWrite(led,LOW);
      }
    }
    else{
      Serial.print("\n  Introduzca la temperatura minima a la que quiere que se accione la alarma: ");
      minima=leer();
      Serial.print(minima, DEC);
      Serial.print(" grados Celsius\n");
      delay(600);
      Serial.print("\n  Introduzca la temperatura maxima a la que quiere que se accione la alarma: ");
      maxima=leer();
      Serial.print(maxima, DEC);
```

```
      Serial.print(" grados Celsius\n");
      delay(600);
      Serial.print("\n  La alarma se accionara fuera del rango: [");
      Serial.print(minima, DEC);
      Serial.print(" C, ");
      Serial.print(maxima, DEC);
      Serial.print(" C]\n\n");
      activa=1;
    }
  break;

  case 4:
    Serial.print("\n\n  [4] Fijar la temperatura del termostato:\n");
    termostato();
  break;
  }
}
```

A continuación procedemos con el análisis de cada una de las funciones a las que se hace referencia desde el 'loop', o como en este caso, desde la función 'setup'. Estamos hablando de la función 'power_on', que tiene como objetivo dejar encendido el módulo GPRS al inicio del programa. Veamos el código correspondiente a la función:

```
void power_on(){
    uint8_t answer=0;
    delay(2000);
    // comprobamos que el modulo esta encendido y responde
    answer = sendATcommand("AT", "OK", 3000);
    if (answer == 0)
    {   // si no responde, lo encendemos
        digitalWrite(onModulePin,HIGH);
        delay(3000);
        digitalWrite(onModulePin,LOW);
        // esperamos a recibir una respuesta del modulo
        while(answer == 0){   // enviamos AT cada dos seg. para ver si responde
            answer = sendATcommand("AT", "OK", 2000);
        }
    }
}
```

Tal como se observa en el código anterior, cada vez que queremos enviar un comando AT a la shield GPRS lo hacemos a través de la función 'sendATcommand'. Esta función recibe como parámetros: el comando que queremos enviar, la respuesta esperada frente a ese comando, y un tiempo límite para obtener esa respuesta antes de dar por supuesto que existe un posible error en la comunicación con el módulo (problemas de cobertura, que el módulo se encuentre apagado, que no se detecte ninguna tarjeta SIM en el módulo, etc.); y devuelve una respuesta (0 o 1) en función del éxito que haya tenido en el envío del comando.

```
int8_t sendATcommand(char* ATcommand, char* expected_answer, unsigned int timeout){
    uint8_t x=0,  answer=0;
    char response[100];
    unsigned long previous;

    memset(response, '\0', 100);    // dejamos la cadena vacia
    delay(100);
```

```
    while( Serial1.available() > 0) Serial1.read();   // limpiamos el buffer de entrada
    Serial1.println(ATcommand);    // enviamos el comando
    x = 0;
    previous = millis();
    // loop esperando la respuesta del modulo
    do{
        // si hay datos en el buffer de entrada, leemos y analizamos la respuesta
        if(Serial1.available() != 0){
            response[x] = Serial1.read();
            x++;
            // verificamos que la respuesta es la esperada
            if (strstr(response, expected_answer) != NULL)
            {
                answer = 1;
            }
        }
    // esperamos respuesta durante un tiempo
    }while((answer == 0) && ((millis() - previous) < timeout));
    return answer;
}
```

La siguiente función se corresponde con el menú que le será mostrado por pantalla al usuario para la elección de las distintas aplicaciones.

```
int menu(){
  int opc;
  Serial.println("* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *");
  Serial.println("*                                                               *");
  Serial.println("*  Elija alguna de las siguientes opciones:                     *");
  Serial.println("*                                                               *");
  Serial.println("*    [1] Consultar temperatura                                  *");
  Serial.println("*    [2] Enviar SMS con el valor de temperatura                 *");
  Serial.println("*    [3] Activar/desactivar alarma por rango de temperatura     *");
  Serial.println("*    [4] Fijar la temperatura del termostato                    *");
  Serial.println("*                                                               *");
  Serial.println("* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *");
  opc=leer();
  while(opc<1 || opc>4){
    Serial.println("\n  Por favor, escoja una opcion correcta.");
    opc=leer();
  }
  return opc;
}
```

A continuación adjuntamos el código correspondiente a la función ‘leer’, a la que se recurre cada vez que queremos leer un dato que ha sido introducido por el usuario a través del Monitor Serial. La función almacena la información recibida en una cadena de caracteres, y acto seguido la convierte en un valor numérico, el cual es devuelto por la función. Puesto que dicha función es la que se mantiene en ejecución de manera continua mientras esperamos ordenes por parte del usuario, en ella se comprueba si la alarma ha sido activada, y en caso afirmativo evalúa si debe accionarse o no. El algoritmo correspondiente al sistema de alarma va incluido en el código de la propia función ‘leer’.

```
int leer (){
  int valor=0;
  char cadena[24];
  byte contador=0;
  float grados;
  memset(cadena, 0, sizeof(cadena));
  while (Serial.available()==0){ //mientras la pantalla esta en espera
    delay(5);
    if(activa==1){ //reviso si tiene que saltar alarma

      grados = Thermistor(analogRead(ThermistorPIN));
      if( (grados>((float)maxima)) || (grados<((float)minima)) ){
        digitalWrite(led,HIGH);
        if(aviso==0){
          aviso=1;
          Serial.print("\n  ALERTA! RANGO DE TEMPERATURA EXCEDIDO\n");
          analogWrite(buzzer,128); //emite sonido
          delay(2500); //espera medio segundo
          digitalWrite(buzzer, LOW); //deja de emitir
          SMS_alarma();
        }
        analogWrite(buzzer,128); //emite sonido
        delay(500); //espera medio segundo
        digitalWrite(buzzer, LOW); //deja de emitir
        delay(500); //espera medio segundo
      }
      else{
        digitalWrite(led,LOW);
      }
    }
  }
  while (Serial.available()>0){ //cuando el usuario escriba, lo leemos
    delay(5);
    cadena[contador]=Serial.read();
    contador++;
  }
  valor=atoi(cadena); //para saber el numero que ha introducido el usuario
  contador=0;
  return valor;
}
```

Comenzamos con las funciones asociadas a la selección de alguna de las opciones mostradas en el menú. La primera de ellas corresponde a la consulta del valor de temperatura. Tras pedir al usuario la frecuencia con la que se mostrará el valor de temperatura por pantalla, es necesario invocar a la función 'calculo_temperatura' para obtener dicho valor y mostrarlo en el Monitor Serial. Adjuntamos el código correspondiente a estas dos funciones:

```
void consultar_temperatura(){
  int frec;
  int periodo;
  Serial.print("\n  Seleccione la frecuencia con la que quiere recibir el valor de temperatura: ");
    frec=leer();
    Serial.print("cada ");
    Serial.print(frec, DEC);
    Serial.println(" segundos.");
    delay(800);
    Serial.println("\n  Pulse [0] para abandonar la consulta.");
    delay(800);
    Serial.println("\n    - Temperatura registrada:\n");
    delay(500);
    periodo=frec*1000; //pasamos a milisegundos
    calculo_temperatura(periodo);
}
```

```
int calculo_temperatura(int x){
    float temp;
    float grados;

    while (Serial.read()!='0'){ //al pulsar '0' saldremos de la funcion
      celsius = Thermistor(analogRead(ThermistorPIN));
      Serial.print("  Celsius: ");
      Serial.print(celsius,1);  // muestra el valor de temperatura en Celsius
      temp = celsius + 273.15;  // convertimos a Kelvin
      Serial.print("\tKelvin: ");
      Serial.print(temp,1);   // muestra el valor de temperatura en Kelvin
      temp = (celsius * 9.0)/ 5.0 + 32.0;  // convertimos a Fahrenheit
      Serial.print("\tFahrenheit: ");
      Serial.print(temp,1);     // muestra el valor de temperatura en Fahrenheit
      Serial.print("\n");
      delay(x);
      if(activa==1)
      { //comprobamos si debe saltar la alarma de temperatura
        grados = Thermistor(analogRead(ThermistorPIN));
        if( (grados>((float)maxima)) || (grados<((float)minima)) ){
          digitalWrite(led,HIGH);
          if(aviso==0){
            aviso=1;
            Serial.print("\n  ALERTA! RANGO DE TEMPERATURA EXCEDIDO\n");
            SMS_alarma();
          }
        }
        else{
          digitalWrite(led,LOW);
        }
      }
   }
   Serial.print("\n  [0] Saliendo. Espere por favor.\n\n");
   delay(800);
}
```

La siguiente función (‘void SMS’) es una de las más interesantes, pues es la encargada del envío de un mensaje de texto con el valor de temperatura detectado en ese mismo momento.

```
void SMS(){
  delay(3000);
  Serial.println("  Conectando a la red...");
  delay(1000);

    while( (sendATcommand("AT+CREG?", "+CREG: 0,1", 1000) ||
            sendATcommand("AT+CREG?", "+CREG: 0,5", 1000)) == 0 );

    Serial.print("  Configurando para modo SMS...");
    delay(1000);
    sendATcommand("AT+CMGF=1", "OK", 1000);    // seleccionamos modo texto
    Serial.println("Enviando el SMS");
    sprintf(aux_string,"AT+CMGS=\"%s\"", phone_number);
    answer = sendATcommand(aux_string, ">", 2000);    // enviamos el numero de movil destino

    if (answer == 1)
    {   //si todo va bien, consultamos el valor de temperatura
        celsius = Thermistor(analogRead(ThermistorPIN));
        Serial1.print("Temperatura = ");  // escribimos la temperatura en el SMS
        Serial1.print(celsius,1);
        Serial1.write(0x1A); // indicamos fin de SMS
        answer = sendATcommand("", "OK", 20000);
```

```
        if (answer == 1){
            Serial.println("  Mensaje enviado");
        }
        else{
            Serial.println("  Error en el envio ");
        }
    }
    else{
       Serial.println("  Error en la comunicación con el modulo: tipo ");
       Serial.println(answer, DEC); //mostramos por pantalla el tipo de error
    }
  delay(2000);
}
```

Para el envío de un SMS como sistema de alerta tras la detección de un valor crítico de temperatura recurrimos a o tra función de similares características a la anterior (void 'SMS_alarma'), que envía un mensaje de texto específico alertando al usuario junto con el valor crítico obtenido en la medida de temperatura, el cual ha hecho saltar el sistema de alarma.

```
void SMS_alarma(){

    while( (sendATcommand("AT+CREG?", "+CREG: 0,1", 1000) ||
            sendATcommand("AT+CREG?", "+CREG: 0,5", 1000)) == 0 );

    sendATcommand("AT+CMGF=1", "OK", 1000);    // seleccionamos modo texto
    Serial.println("Enviando SMS de alerta por temperatura extrema");
    sprintf(aux_string,"AT+CMGS=\"%s\"", "6[illegible]");
    answer = sendATcommand(aux_string, ">", 2000);    // enviamos el numero de movil destino

    if (answer == 1){   //si todo va bien, consultamos el valor de temperatura
        celsius = Thermistor(analogRead(ThermistorPIN));
        // escribimos el contenido del SMS junto con el valor de temperatura
        Serial1.print("ALERTA! RANGO DE TEMPERATURA EXCEDIDO. Temperatura = ");
        Serial1.print(celsius,1);
        Serial1.write(0x1A); // indicamos fin de SMS
        answer = sendATcommand("", "OK", 20000);
        if (answer == 1){
            Serial.println("  Mensaje enviado");
        }
        else{
            Serial.println("  Error en el envio ");
        }
    }
    else{
       Serial.println("  Error en la comunicación con el modulo: tipo ");
       Serial.println(answer, DEC); //mostramos por pantalla el tipo de error
    }
}
```

Llegamos a la última de las funciones (void 'termostato'), correspondiente a la configuración del termostato en función de los grados centígrados a los que el usuario quiere regular el mismo. Dicho valor será introducido por el usuario tras la selección de la opción correspondiente al accionamiento del termostato desde el menú de inicio. A continuación adjuntamos el código correspondiente a la función termostato:

```
void termostato()
{
  int rotar;

  Serial.print("\n  Introduzca la temperatura a la que quiere programar el termostato: ");
  rotar=leer();
  Serial.println(rotar, DEC);

  while(rotar<16 || rotar>25)
  {
    Serial.print("\n  Fuera de rango. Por favor introduzca un valor correcto [entre 16 y 25 C] \n ");
    Serial.print("\n  Introduzca la temperatura a la que quiere programar el termostato: ");
    rotar=leer();
    Serial.print(rotar, DEC);
  }

  //algoritmo para ajustar los grados con el angulo de rotacion necesario
  angulo=(25-rotar)*20;
  miServo.write(angulo);
  delay(1000);

  Serial.print("\n  Valor de temperatura fijado a ");
  Serial.print(rotar, DEC);
  Serial.println(" C\n ");
  delay(1000);
}
```

Llegados a este punto ya se han adjuntado cada uno de los fragmentos en los que hemos dividido el código de la aplicación principal, el cual hemos ido argumentando. Ahora es el turno de poner a prueba el sistema y evaluar el correcto funcionamiento de este programa, el cual analizaremos detenidamente en el próximo apartado.

4.7.2. Evaluación del sistema de gestión local

En este apartado evaluaremos paso a paso el funcionamiento del programa correspondiente al sistema de control de temperatura de gestión local, a través del interfaz Monitor Serial de Arduino. Se adjuntarán varias capturas de pantalla tomadas durante la ejecución del mismo con el fin de contribuir a una mejor comprensión de las utilidades de la aplicación que ha sido diseñada.

En primer lugar se adjunta una fotografía con el resultado del montaje del sistema al completo. Como vemos en la imagen correspondiente a la figura 4.39, el montaje se lleva a cabo con la ayuda de una placa de pruebas (‘Protoboard’), aunque la idea es fabricar una pequeña placa de circuito impreso en la que integrar los distintos componentes y eliminar el siempre antiestético cableado, reduciendo el tamaño del sistema, dotándole de otro aspecto, y en definitiva, convirtiendo al sistema en una única plataforma de diseño modular más compacta y robusta.

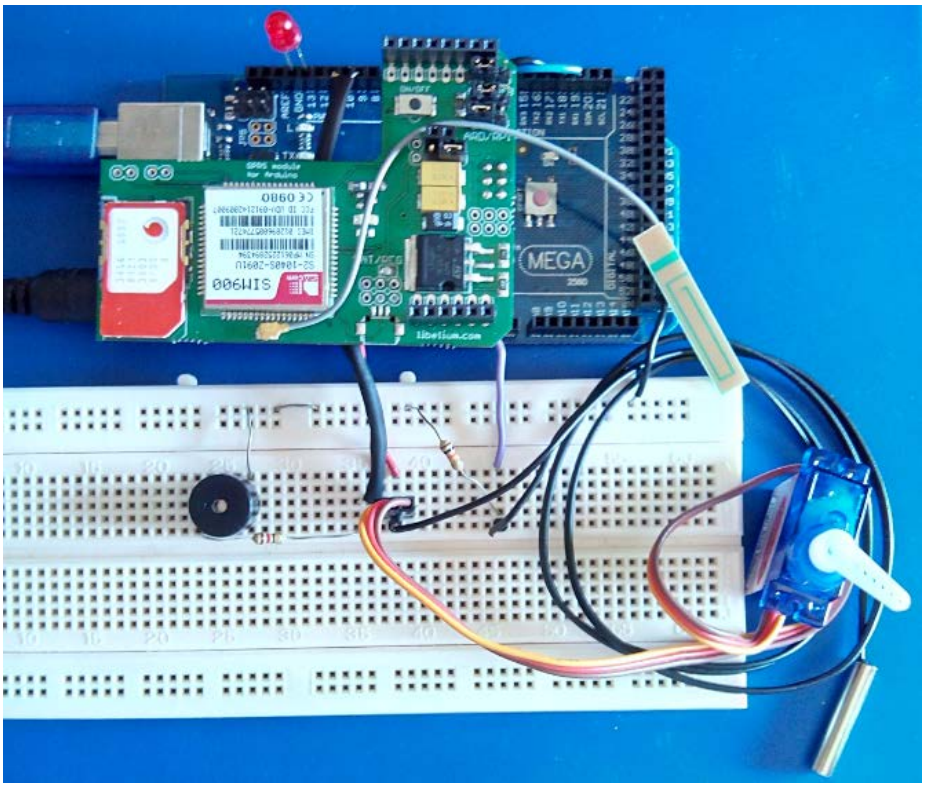


Figura 4.39: Montaje correspondiente al prototipo del sistema completo

A continuación conectamos el sistema al puerto USB del PC, y procedemos con la carga del programa (no debemos olvidar colocar los jumpers en modo "gateway" durante el proceso de carga). Una vez que el programa ha sido cargado correctamente lo desconectamos del puerto USB, y cambiamos la posición de los jumpers a modo "Arduino". Acto seguido, conectamos de nuevo al PC y enchufamos también la fuente de alimentación. Tras abrir el Monitor Serial el programa procederá con su ejecución.

Lo primero que aparece por pantalla es el menú de inicio, tal como se muestra en la siguiente figura 4.40.

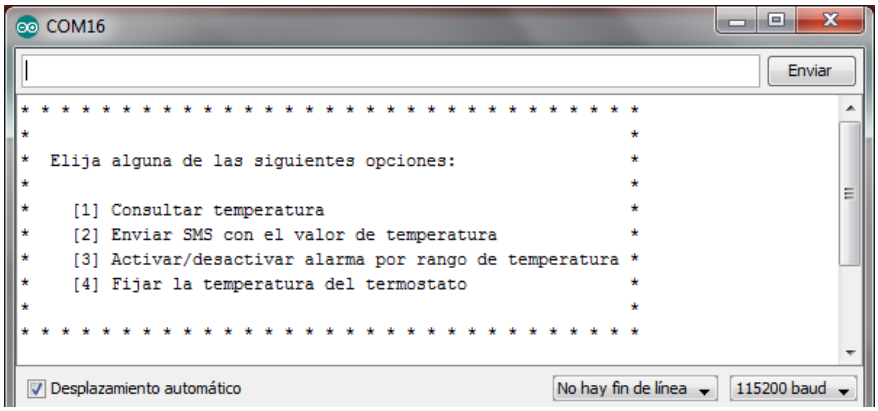


Figura 4.40: Menú de inicio

El usuario escogerá una de las opciones tecleando desde el número correspondiente. Tras finalizar cada una de ellas se volverá a mostrar el menú, y el programa se mantendrá a la espera de una nueva orden por parte del usuario.

A continuación vamos a ir viendo lo que hace cada una de las opciones. Comenzamos por la primera de ellas, consultar el valor de temperatura. En la figura 4.41 vemos la captura de pantalla correspondiente a esta opción. Se le pide al usuario que introduzca la frecuencia con la que quiere que se muestre por pantalla el valor de temperatura, y tras ello, comienza a listar los valores de temperatura obetenidos.

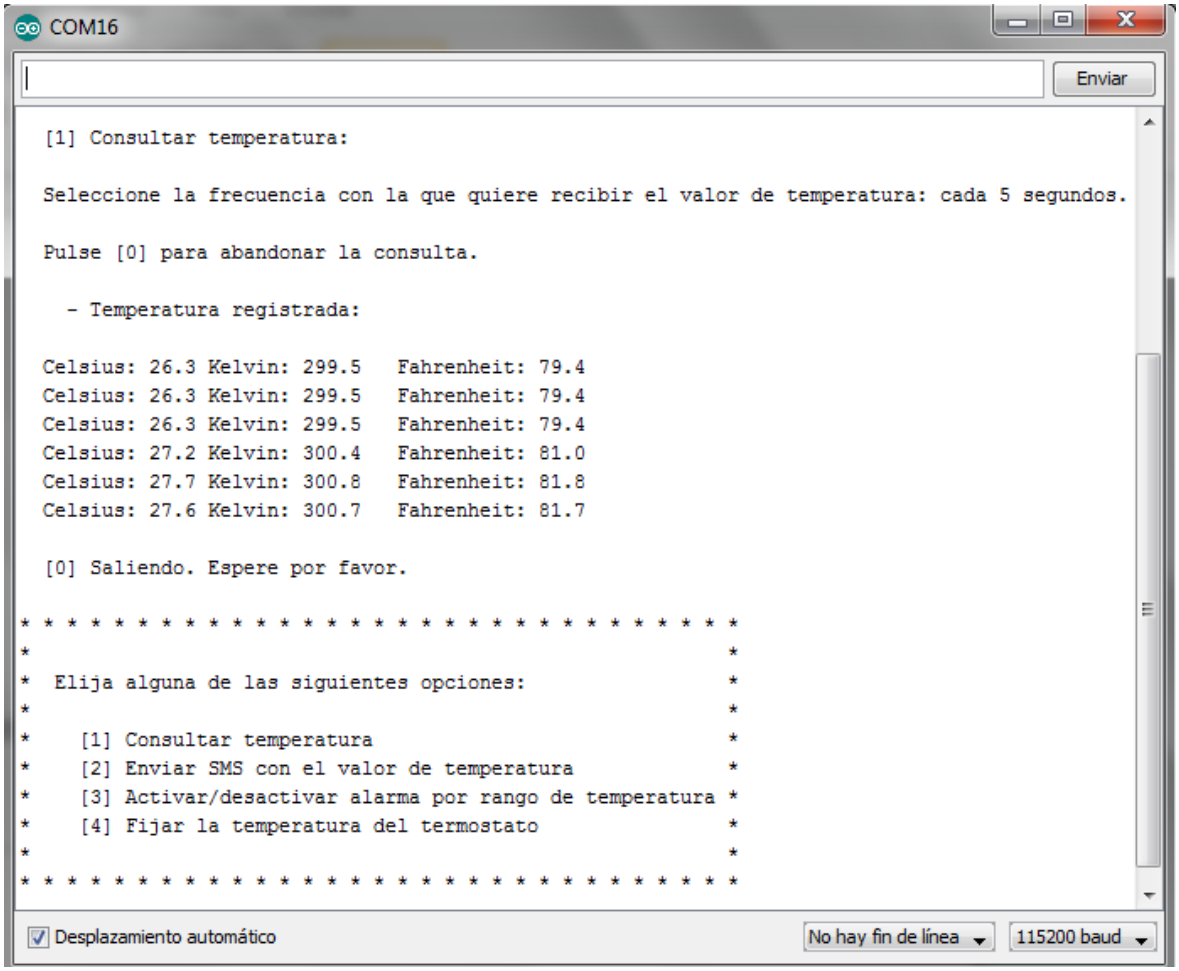


Figura 4.41: Consultar el valor de temperatura

La intención de llevar a cabo la consulta de este modo surge, como siempre, con vistas a una mejora de la aplicación en la segunda fase de este proyecto, con la idea de contar con un portal web al que ir subiendo cada cierto tiempo los valores de temperatura registrados a lo largo del día, incluso poder generar gráficas en función de los cambios en la temperatura ambiente.

La siguiente opción consiste en el envío del valor de temperatura a través de un mensaje de texto. Para ello, el programa solicita por pantalla el número de móvil al que se quiere enviar el valor de temperatura, tal y como se muestra en la figura 4.42. En la mejora del proyecto que llevará a cabo mi compañero, en la que contaremos con una aplicación Android para la gestión de la aplicación, podremos almacenar algunos números de teléfono favoritos entre los que podremos seleccionar al destinatario de los SMS sin necesidad de tener que introducir manualmente el número, incluso seleccionarlo directamente a partir de los contactos de la tarjeta SIM.

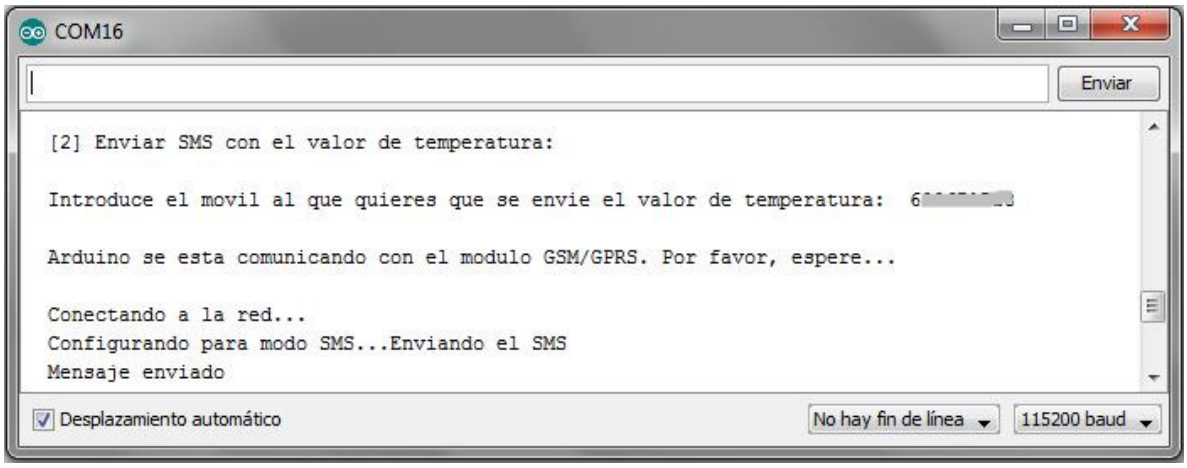


Figura 4.42: Enviar SMS con el valor de temperatura

En la figura 4.43 podemos ver el mensaje recibido en el teléfono móvil con el valor de temperatura.

Figura 4.43: SMS recibido con el valor de temperatura

La opción número 3 asume las funciones de activar y desactivar el sistema de alarma. Cuando se active la alarma el usuario deberá imponer unos márgenes que delimiten el rango de temperatura en la que la alarma no debe accionarse. Solo se accionará cuando el valor de temperatura detectado exceda los límites impuestos por el usuario. Además de activarse una

serie de actuadores de alerta (en este caso un LED y un pequeño buzzer), se enviará al usuario un SMS alertándole de la situación, en el que quedará además reflejado el valor de temperatura extremo obtenido en ese instante. En la figura 4.43 vemos el resultado de lo que aparece por pantalla cuando activamos la alarma y también cuando forzamos el accionamiento de la misma calentando el sensor de temperatura.

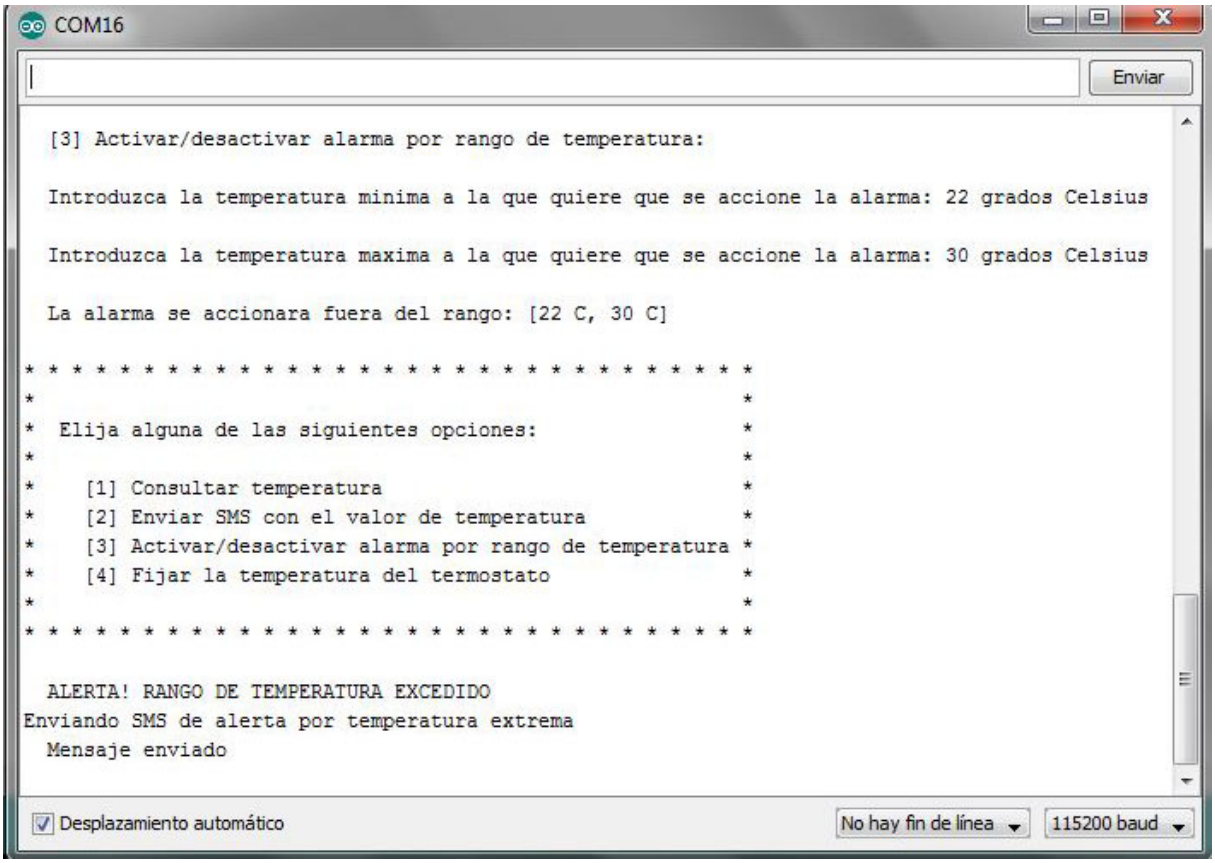


Figura 4.44: Activar alarma por temperatura extrema

El mensaje de texto que recibimos en el terminal móvil como consecuencia de la activación de la alarma es idéntico al que se muestra a continuación en la figura 4.44.

Figura 4.45: SMS recibido tras el accionamiento del sistema de alarma

A continuación se muestra también la captura de pantalla correspondiente a la desactivación de la alarma. Para ello seleccionaremos la opción 3, y seguiremos las instrucciones que se indican por pantalla para que el sistema de alarma quede desactivado. Veámoslo en la siguiente imagen:

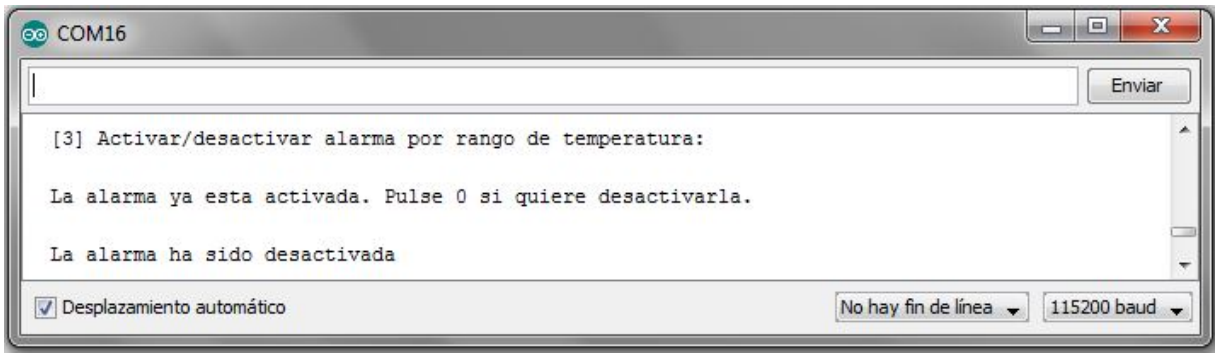


Figura 4.46: Desactivar el sistema de alarma

Por último, se adjunta el resultado de la función correspondiente a la regulación del termostato. Como podemos ver en la figura 4.47, el usuario será quién introduzca el valor de temperatura al que quiere fijar el termostato, y el programa actuará en consecuencia haciendo girar el servo y por consiguiente la rueda de un posible termostato.

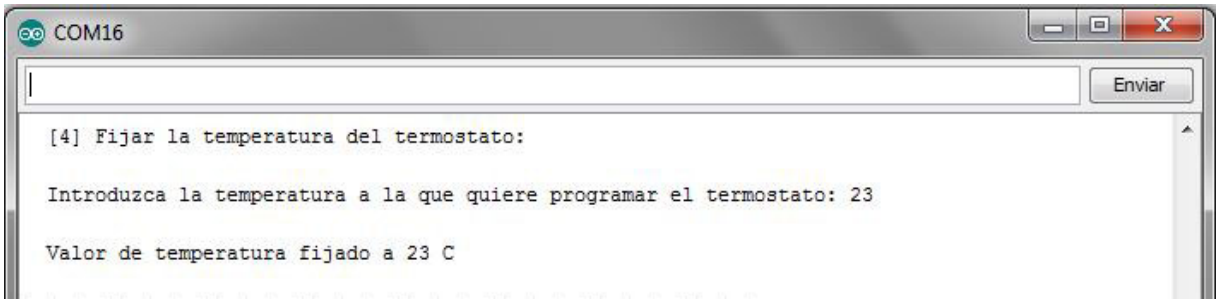


Figura 4.47: Fijar la temperatura del termostato

4.7.3. Código de aplicación para la gestión remota de la temperatura. Modo 'standalone'

Entramos en detalle con la segunda de las aplicaciones que se ha diseñado en este proyecto. Se trata de un programa que convierte a nuestro sistema en una plataforma desatendida (que solo necesita estar conectada a la fuente de alimentación) con el fin de ser instalada en una vivienda u otro tipo de local y controlar a distancia su temperatura.

La gestión remota se llevará a cabo mediante el intercambio de mensajes de texto (SMS) entre un teléfono móvil y el módulo GPRS/GSM que forma parte de nuestro sistema. En la segunda fase del proyecto, mi compañero hará posible está comunicación a través de internet, en lugar de por SMS como se ha implementado en esta primera fase, pero la intención era demostrar que podemos controlar remotamente nuestro sistema aprovechando los sistemas de comunicaciones móviles.

Pasamos pues a presentar el diagrama de flujo correspondiente a esta aplicación, el cual se adjunta en la figura 4.48.

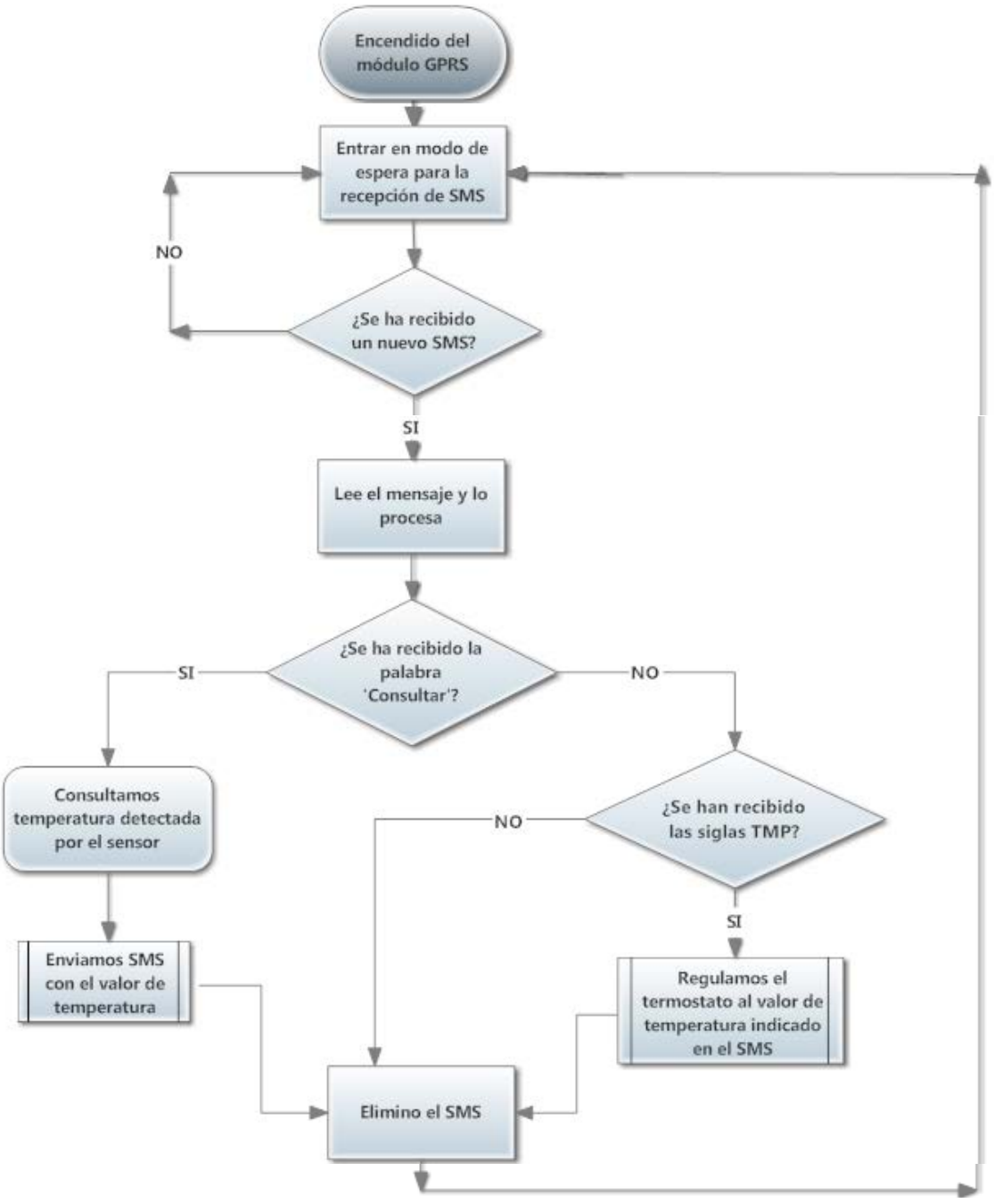


Figura 4.48: Diagrama de flujo correspondiente al sistema de gestión remota

A partir del diagrama de flujo anterior procedemos con el diseño de la aplicación. Puesto que la mayoría de las funciones son las mimas que hemos utilizado en el código general de la aplicación (salvo pequeñas modificaciones), adjuntamos directamente el resultado del sketch, en el que se incluyen algunos comentarios sobre las líneas que conviene resaltar.

```
#include <math.h>
#include <Servo.h>

#define ThermistorPIN 0  // Pin analogico 0

float vcc = 4.91;  // valro de tension continua
float pad = 9850;  // valor real de resistencia
float thermr = 10000; // valor nominal de la resistencia
float celsius;

Servo miServo;
int8_t answer;
int x;
int j;
char phone_number[10]; //variable para guardar el numero de movil
char aux_string[30]; //variable para guardar una cadena de caracteres mientras es procesada
char mensaje[200]; //variable para guardar el texto del mensaje
int angulo=90; //eje central del servo
int led = 13; // pin del led
int onModulePin = 2;  // pin correspondiente a boton encendido
float Thermistor(int RawADC) {
  //algoritmo para el calculo de la temperatura
  long Resistance;
  float Temp;
  Resistance=((1024 * pad / RawADC) - pad);
  Temp = log(Resistance);
  Temp = 1 / (0.001129148 + (0.000234125 * Temp) +
         (0.0000000876741 * Temp * Temp * Temp));
  Temp = Temp - 273.15;
  return Temp;    // Devolver el valor de temperatura en celsius
}
void setup(){
  pinMode(onModulePin, OUTPUT);
  Serial.begin(115200); //baudrate del puerto serie con PC
  Serial1.begin(115200); //baudrate del puerto comunicacion con modulo GPRS
  pinMode(led,OUTPUT);
  power_on(); //encendemos el modulo GPRS
  miServo.attach(9);
  Serial.println("  Conectando a la red...");
  delay(4000);
  while( (sendATcommand("AT+CREG?", "+CREG: 0,1", 1000) ||
            sendATcommand("AT+CREG?", "+CREG: 0,5", 1000)) == 0 );
  for(j=0; j<3; j++){
    digitalWrite(led,HIGH);  // Enciende el LED
    delay(500);              // Temporiza medio segundo (500ms)
    digitalWrite(led,LOW);   // Apaga el LED
    delay(300);     // Temporiza 300ms
  }
  Serial.println("Setting SMS mode...");
  sendATcommand("AT+CMGF=1", "OK", 1000);    // modo texto
  sendATcommand("AT+CPMS=\"SM\",\"SM\",\"SM\"", "OK", 1000);  // memoria de la SIM
  sendATcommand("AT+CMGD=1,4", "OK", 1000);
}
```

```
void power_on(){
    uint8_t answer=0;
    delay(4000);
    // comprobamos que el modulo esta encendido y responde
    answer = sendATcommand("AT", "OK", 3000);
    if (answer == 0)
    {   // si no responde, lo encendemos
        digitalWrite(onModulePin,HIGH);
        delay(3000);
        digitalWrite(onModulePin,LOW);
        // esperamos a recibir una respuesta del modulo
        while(answer == 0){   // enviamos AT cada dos seg. para ver si responde
            answer = sendATcommand("AT", "OK", 2000);
        }
    }
}

int8_t sendATcommand(char* ATcommand, char* expected_answer, unsigned int timeout){
    uint8_t x=0,  answer=0;
    char response[100];
    unsigned long previous;

    memset(response, '\0', 100);    // dejamos la cadena vacia
    delay(100);
    while( Serial1.available() > 0) Serial1.read();   // limpiamos el buffer de entrada
    Serial1.println(ATcommand);    // enviamos el comando
    x = 0;
    previous = millis();
    // loop esperando la respuesta del modulo
    do{
        // si hay datos en el buffer de entrada, leemos y analizamos la respuesta
        if(Serial1.available() != 0){
            response[x] = Serial1.read();
            x++;
            // verificamos que la respuesta es la esperada
            if (strstr(response, expected_answer) != NULL)
            {
                answer = 1;
            }
        }
    // esperamos respuesta durante un tiempo
    }while((answer == 0) && ((millis() - previous) < timeout));
    Serial.println(response);
    return answer;
}


int8_t recibeATcommand(char* expected_recibir, unsigned int timeout){
    uint8_t x=0,  answer=0;
    char response[100];
    unsigned long previous;
    memset(response, '\0', 100);    // dejamos la cadena vacia
    delay(100);
    while( Serial1.available() > 0) Serial1.read();   // limpiamos el buffer de entrada
    x = 0;
    previous = millis();
    // loop esperando la respuesta del modulo
    do{
        // si hay datos en el buffer de entrada, leemos y analizamos la respuesta
```

```
        if(Serial1.available() != 0){
            response[x] = Serial1.read();
            x++;
            // verificamos que la respuesta es la esperada
            if (strstr(response, expected_recibir) != NULL)
            {
                answer = 1;
            }
        }
    // esperamos respuesta durante un tiempo
    }while((answer == 0) && ((millis() - previous) < timeout));
    return answer;
}

void SMS(){
    sendATcommand("AT+CMGF=1", "OK", 1000);    // seleccionamos modo texto
    sprintf(aux_string,"AT+CMGS=\"%s\"", phone_number); //pasamos el numero de movil del remitente
    answer = sendATcommand(aux_string, ">", 2000);

    if (answer == 1)
    {   //si todo va bien, consultamos el valor de temperatura
        celsius = Thermistor(analogRead(ThermistorPIN));
        Serial1.print("Temperatura = ");  // escribimos la temperatura en el SMS
        Serial1.print(celsius,1);
        Serial1.write(0x1A); // indicamos fin de SMS
        answer = sendATcommand("", "OK", 20000);
        if (answer == 1){
            Serial.println("  Mensaje enviado");
        }
        else{
            Serial.println("  Error en el envio ");
        }
    }
    else{
       Serial.println("  Error en la comunicación con el modulo: tipo ");
       Serial.println(answer, DEC); //mostramos por pantalla el tipo de error
    }
  delay(2000);
}

void termostato(int n){
  if(n<16 || n>25)
  {
    Serial.print("\n  Fuera de rango. \n ");
  }
  else{
    //algoritmo para ajustar los grados con el angulo de rotacion necesario
    angulo=(25-n)*20;
    miServo.write(angulo);
  }
}


void loop(){
  int respuesta=0;
  int consulta=0;
  int compruebo=0;

  //Esperando recibir un SMS, indicado por el comando +CMTI: SM,1
  do{
    respuesta = recibeATcommand("+CMTI: \"SM\",1", 2000);
```

```
    if(respuesta == 1){ // si recibimos procedemos con su lectura
      Serial.println("  SMS Recibido");
      compruebo=sendATcommand("AT+CMGR=1", "+CMGR:", 2000);
      if (compruebo == 1){
          compruebo = 0;
          memset(mensaje, '\0', 200);
          while(Serial1.available() == 0);
          x=0;
          // en este loop leemos el contenido del SMS
          do{
              // mientras halla datos sin leer sigo leyendo el mensaje
              if(Serial1.available() > 0){
                  mensaje[x] = Serial1.read();
                  x++;
                  // buscamos el OK de fin de SMS
                  if (strstr(mensaje, "OK") != NULL)
                  {
                      compruebo = 1;
                  }
              }
          }while(compruebo == 0);
          mensaje[x] = '\0';     //Ya tenemos el mensaje
          Serial.print(mensaje);
      }
      else
      {
          Serial.print("Error en la recepcion del SMS");
          Serial.println(compruebo, DEC);
      }

      // comenzamos a procesar el contenido del SMS
      if(strstr(mensaje, "Consultar") != NULL){ // si pone la palabra consultar...
        String valorT (mensaje);
        int posT = valorT.indexOf( "+34" );
        valorT = valorT.substring( posT+3,posT+12 );
        //char num_usuario[12];
        valorT.toCharArray(phone_number, sizeof(phone_number));
        Serial.print(" \nEnviando valor de temperatura por SMS al numero: ");
        Serial.println( phone_number );
        SMS();    //utilizamos funcion SMS para enviar mensaje
        sendATcommand("AT+CMGD=1", "OK", 2000); //eliminar el SMS para esperar otro
      }
      else{
        if(strstr(mensaje, "TMP") != NULL){ //si pone las siglas TMP....
          String valorT (mensaje);
          int posT = valorT.indexOf( "TMP" );
          valorT = valorT.substring( posT+3,posT+5 );
          char carray[5];
          valorT.toCharArray(carray, sizeof(carray));
          Serial.println( carray );
          int n = atoi(carray);
          Serial.println( n ); //ya tengo el entero para mover el servo en consecuencia
          termostato(n);
          sendATcommand("AT+CMGD=1", "OK", 2000); //eliminar el SMS para esperar otro
        }
        else{ // si no es ni consultar ni TMP, entonces lo ignoro
          sendATcommand("AT+CMGD=1", "OK", 2000);
        }
      }
    }
  }while(respuesta == 0); //modo espera para recepcion de SMS
}
```

4.7.4. Evaluación del sistema de gestión remota

De la misma manera que evaluamos anteriormente el funcionamiento del programa principal, ahora es el turno de analizar el rendimiento de esta otra aplicación. Para ello cargaremos el código del programa sobre nuestra plataforma Arduino, manteniendo exactamente el mismo esquema de montaje que vimos en el apartado correspondiente al sistema de gestión local (Figura 4.39).

A diferencia del programa principal, en este no será necesaria la conexión al PC para que el programa permanezca en ejecución, de hecho, el principal objetivo de esta segunda aplicación es disponer de una sistema de gestión remoto mientras este permanece trabajando en modo 'standalone' (instalación desatendida).

Una vez que el código ha sido cargado satisfactoriamente solo tenemos que alimentar el sistema para que éste entre en funcionamiento. Tal y como se muestra en la figura 4.49, conectamos la fuente de alimentación a la plataforma, y esperamos unos segundos hasta que veamos parpadear el LED en tres ocasiones consecutivas, lo que será indicativo de que el sistema ya se encuentra operativo y a la espera de recibir órdenes vía SMS.

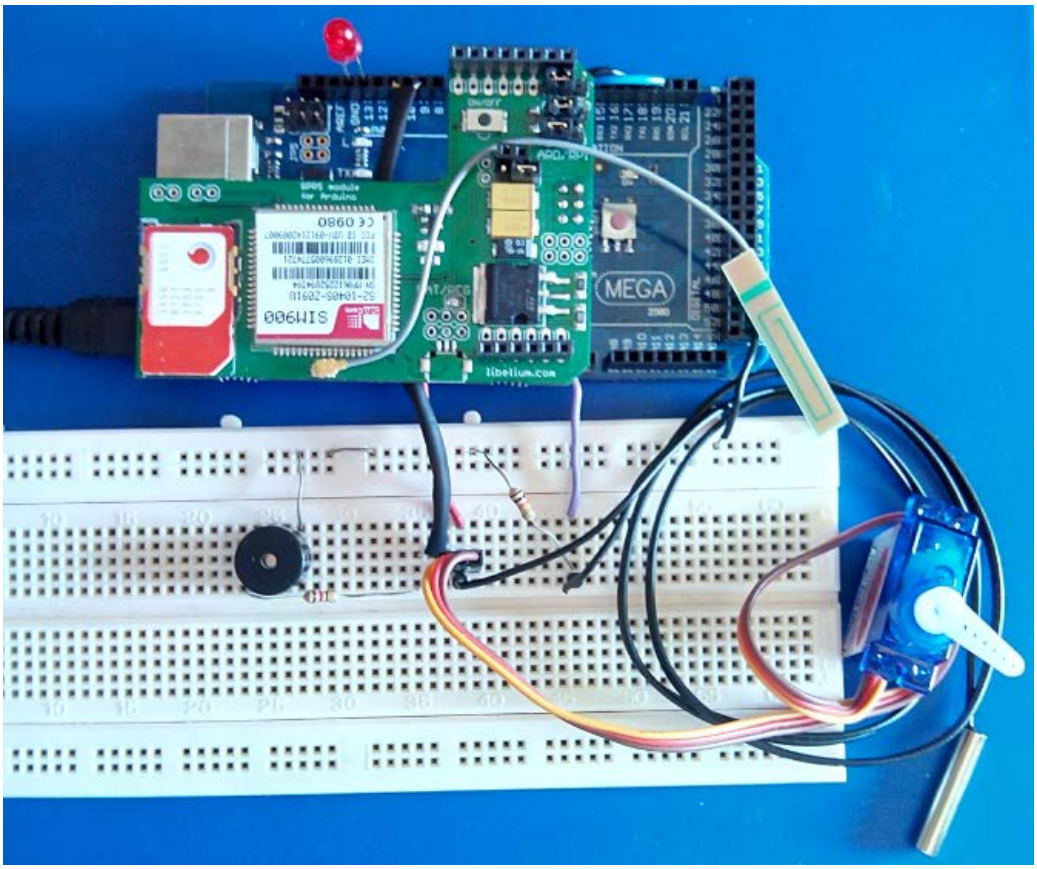


Figura 4.49: Montaje del sistema en modo 'standalone'

A continuación enviaremos los mensajes de texto para ver si el sistema responde como es debido ante nuestras instrucciones. Recordemos que el programa deber ser capaz de interpretar dos tipos de mensaje, los que incluyan la palabra “Consultar” y los que contengan las siglas “TMP” seguido del valor temperatura. En el primero de los casos (“Consultar”) responderá al remitente con un SMS informándole del valor de temperatura detectado en ese mismo momento; y respecto al segundo (“TMP<valor>”) accionará el motor servo con la intención de regular un posible termostato en función del correspondiente valor de temperatura que el usuario le ha indicado a través del mensaje.

Para contribuir a una mejor comprensión del funcionamiento del sistema se van a ir adjuntando diferentes capturas de pantalla correspondientes a los SMS intercambiados durante la comunicación con la plataforma, e iremos comentando las acciones que el programa ha llevado a cabo en función de ese intercambio de mensajes de texto.

En primer lugar vamos a hacer la prueba para el primero de los casos. Enviamos por tanto un SMS con la palabra “Consultar”. Tras su recepción Arduino procesará el contenido del mensaje y actuará en consecuencia, enviando un SMS de respuesta con el valor actual de temperatura. En la figura 4.50 podemos ver el intercambio de mensajes.

Figura 4.50: Consultar temperatura por SMS

Como vemos en la imagen correspondiente a la figura 4.50, la consulta del valor de temperatura a través de mensajes de texto funciona perfectamente. También llevamos a cabo la prueba de enviar un SMS con una palabra desconocida para asegurarnos de que el dispositivo la obviase y eliminase el mensaje, y como era de esperar, así resultó.

Para terminar pondremos a prueba el sistema ante mensajes que incluyan las siglas “TMP” seguidas del valor de temperatura al que se pretende regular el termostato. Se envía por tanto un SMS con la intención de hacer girar el motor servo hasta la posición correspondiente a los 16 grados centígrados, tal como vemos en el mensaje adjunto en la figura 4.51.

Figura 4.51: Ajustar la temperatura del termostato por SMS

Tras la recepción del mensaje, tal y como cabía esperar, Arduino acciona el servomotor haciéndolo girar hasta la posición equivalente a esos 16 grados centígrados que indicamos en el mensaje, quedando así confirmado el correcto funcionamiento de esta segunda aplicación para la gestión remota de la temperatura.

5. CONCLUSIONES

Tras la conclusión de esta primera fase del proyecto correspondiente a un sistema de control de la temperatura, hacemos balance entre los objetivos que nos planteamos al inicio del mismo y los que se han llegado a conseguir.

Como ya se ha comentado a lo largo del proyecto, se trataba de un trabajo conjunto en el que dos compañeros pretendían desarrollar un sistema más completo. Al tener que separar el proyecto en dos partes, los objetivos finales también deben compartirse, y la intención es alcanzar nuestras metas de inicio tras el desarrollo de ambas fases del proyecto.

En esta primera parte, el objetivo principal siempre fue el de construir una pequeña plataforma de pruebas sobre la que analizar las distintas funcionalidades que podría ofrecernos el equipamiento adquirido (Arduino, shield GPRS, sensor de temperatura y varios actuadores). Tras finalizar el proyecto se puede afirmar que dicho propósito ha sido conseguido con éxito, pues ahora tenemos claro cómo podemos comunicarnos a través del módulo GPRS/GSM e interactuar con varios sensores y actuadores, todo ello conectado sobre la plataforma Arduino, la cual también hemos aprendido a dominar.

Por otro lado, una de las principales intenciones del proyecto era construir un dispositivo de bajo coste que pudiese estar alcance prácticamente de cualquier consumidor. Si analizamos el presupuesto total de este primer sistema, hemos logrado mantenernos entre los límites teóricos que nos habíamos fijado desde un primer momento, donde se impuso no superar la barrera de los 120€. En la figura 5.1 pode mos ver el desglose del presupuesto total del sistema.

DESCRIPCION	Uds.	PVP unit	Subtotales
Arduino MEGA	1	23,45 €	**23,45 €**
GPRS/GSM Quadband Module for Arduino (SIM900)	1	76,23 €	**76,23 €**
Tarjeta SIM Vodafone	1	- €	**0,00 €**
Sensor de temperatura (SEN118A2B)	1	1,82 €	**1,82 €**
Micro Servo 9G	1	4,49 €	**4,49 €**
Buzzer Activo 5V (Zumbador magnético)	1	1,25 €	**1,25 €**
LED, resistencias y otros accesorios	1	0,90 €	**0,90 €**
		TOTAL =	**108,14 €**

Figura 5.1: Presupuesto total del sistema

Para terminar, me gustaría hacer hincapié en las posibilidades de mejora del sistema que quedan pendientes para la segunda fase del proyecto. En ella se desarrollará una aplicación basada en Android, para móvil o Tablet, con el propósito de hacer más dinámica la gestión y control del sistema por parte de un posible usuario final. Consideramos que se trata de un concepto prioritario en este tipo de sistemas que requieren de una manipulación para mantener el dispositivo en funcionamiento, pues un usuario medio rechazará tener que memorizar varios tipos de SMS para dar distintas instrucciones al sistema. Esto quedará resuelto con el desarrollo de la aplicación Android, gracias al cual, el usuario podrá contar con un sencillo e intuitivo interfaz para el control del dispositivo.

6. ANEXO_A: REFERENCIAS

[1] http://www.ngeeks.com/la-cobertura-de-adsl-en-espana/ (última consulta Julio 2013)

[2] http://blogcmt.com/2012/10/25/mapa-de-la-cobertura-3g-en-espana/ (última consulta Julio 2013)

[3] https://xively.com/ (última consulta Julio 2013)

[4] http://www.kurzweilai.net/ibm-open-sources-internet-of-things-protocol (última consulta Julio 2013)

[5] http://es.wikipedia.org/wiki/Sistema_global_para_las_comunicaciones_m%C3%B3viles (última consulta Julio 2013)

[6] http://ocw.upm.es/teoria-de-la-senal-y-comunicaciones-1/comunicaciones-moviles-digitales/contenidos/Presentaciones/GSM-07.pdf (última consulta Julio 2013)

[7] http://es.wikipedia.org/wiki/Servicio_general_de_paquetes_v%C3%ADa_radio (última consulta Julio 2013)

[8] http://www.cooking-hacks.com/skin/frontend/default/cooking/pdf/SIM900_AT_Command_Manual.pdf (última consulta Julio 2013)

[9] http://www.arduino.cc/es/ (última consulta Agosto 2013)

[10] http://arduino.cc/es/Main/Software (última consulta Agosto 2013)

[11] http://botscience.wordpress.com/2012/06/05/historia-de-arduino-y-su-nacimiento/ (última consulta Agosto 2013)

[12] http://arduino.cc/es/Guide/Environment (última consulta Agosto 2013)

[13] http://www.cooking-hacks.com/index.php/catalogsearch/result/?q=gprs (última consulta Agosto 2013)

[14] http://arduino.cc/es/Main/Software (última consulta Agosto 2013)

[15] http://arduino.cc/es/Tutorial/Blink?from=Tutorial.BlinkingLED (última consulta Agosto 2013)

[16] http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Temp/TMP35_36_37.pdf (última consulta Agosto 2013)

[17] http://learn.adafruit.com/tmp36-temperature-sensor/using-a-temp-sensor (última consulta Agosto 2013)

[18] http://www.cooking-hacks.com/index.php/documentation/tutorials/arduino-gprs-gsm-quadband-sim900 (última consulta Agosto 2013)